Belegarbeit mit dem Thema:

Geometrische Analyse und optimierte Kabelführung bei der Rechnernetze-Projektierung

Integration vorhandener Projektierungstools in die CANDY[1]-Umgebung

Philipp Bönisch
Matr.-Nr.: 2839385
Fakultät Informatik
TU Dresden

31.01.2006

[1]Computer Aided Network Design UtilitY

Zusammenfassung

Dieser Beleg ist im Rahmen meiner Belegarbeit an dem Lehrstuhl für Rechnernetze, Fakultät Informatik an der TU Dresden unter der Betreuung von Dr. rer. nat. D. Gütter und Dr.-Ing. A. Luntovskyy entstanden. Ziel des Belegs ist es, das an dem Lehrstuhl entwickelte Programm CANDY um eine weitere Komponente zu erweitern. Diese soll die Aufgabe haben, eine optimale Kabelführung in einem Netz aus Kabelkanälen zu berechnen und auszugeben.

Technische Universität Dresden / Fakultät Informatik

AUFGABENSTELLUNG FÜR DIE BELEGARBEIT

Name, Vorname: Bönisch, Phillipp

Studiengang: Informatik

Matr. Nr.: 2839385

Email: boenisch.p@web.de

Thema: *Geometrische Analyse und optimierte Kabelführung bei RN-Projektierung; Integration vorhandener Projektierungstools in die CANDY-Umgebung*

Zielstellung:

Für die Projektierung von Rechnernetzen ist die Optimierung der Kabelführung mit Berücksichtigung der Konzepte strukturierter Verkabelung der großen Bedeutung.

Für das Tertiärbereich der Verkabelung eines LANs sollte ein parametrisiertes 2D-Modell entwickelt werden. Die Arbeit muss Optimierungsaspekte bei Leitungsführung betrachten. Die Optimierungskriterien z.B. wären: Kabellänge, Anzahl von Koppler, Verteiler und Verstärker, Anzahl von Löchern und Übergänge, Gesamtkosten bei Leitungsführung etc. In der Arbeit muss man auf existierende Optimierungsalgorithmen mit Nutzung des 2D-Modells angehen (Lee Wave Algorithm, Bellman's Dynamic Programming Algorithm etc.)

Im Rahmen der Arbeit soll auch ein XML-basiertes Beschreibungsmodell für die Gebäudegeometrie bzw. die Kabeltrassen in Rechnernetzen erarbeitet werden. Als Vorlage ist die existierende problem-orientierte Sprache NDMLv1.0 (Network Design Markup Language) zu verwenden. Das oben genannte Modell muss in diese Sprache abgebildet werden. In der Belegarbeit muss man definitiv zeigen, dass die abstrakten Datentypen, welche durch NDML gemappt werden, optimal zu den CANDY-spezifischen Zwecken sind.

Weiterhin ist innerhalb des CANDY-Projekts eine Schnittstelle des Modells zu weiteren Projektierungs-Tools bzw. als Webservices mit Nutzung des J2EE-Framework zu konzipieren und ein Trassierungsprogramm für strukturierte Verkabelung mit dem Schwerpunkt „Tertiärsysteme" zu implementieren. Das Modell dient als Grundlage (semantisches Schema) zum Aufbau entsprechender Datenbank und muss in normalisierter Form dargestellt werden bzw. sind die abstrakten Datentypen und Objekte (Komponenten, Services) optimiert.

Weiterhin sind innerhalb der Arbeit die Konzepte weiterer Integration vorhandener Projektierungstools als Webservices in die CANDY-Umgebung zu entwickeln bzw. mit Nutzung des J2EE-Framework.

Schwerpunkte:

- Recherche und Analyse bestehender Algorithmen zur Optimierung der Kabelführung (Lee Wave Algorithm, Bellman's Dynamic Programming Algorithm etc.)
- Analyse der im Rahmen des CANDY-Projekts benutzten XML-basierten Dokumenten- und –Verarbeitungsmodelle
- Konzipieren und Entwicklung des Trassierungsprogramms (bzw. als Webservices mit Nutzung des J2EE-Framework)
- Testen anhand einer Beispielnetzwerkgeometrie
- Entwicklung der Konzepte weiterer Integration vorhandener Projektierungstools in die CANDY-Umgebung bzw. mit Nutzung des J2EE-Framework als Webservices.

Betreuer:	Dr. Dietbert Gütter Dr. Andriy Luntovskyy
Verantwortlicher Hochschullehrer:	Prof. Dr. A. Schill
Institut:	Institut für Systemarchitektur
Lehrstuhl:	Rechnernetze
Beginn am:	1. April 2005
Einzureichen am:	1. Oktober 2005

Inhaltsverzeichnis

1 Einführung **5**

1.1 Erläuterung der Aufgabenstellung 5

2 Vorarbeit **5**

2.1 Analyse der Realität 6

2.1.1 Der Grundriss 6

2.1.2 Der Kabelkanal 6

2.2 Analyse vorhandener Algorithmen 7

2.2.1 Bellman-Ford-Algorithmus 7

2.2.2 Dijkstra-Algorithmus 8

2.2.3 Resümee für die Implementierung 9

2.3 Analyse der NDML 11

2.4 Gesetzliche Bestimmungen und Normen 13

2.4.1 Graphische Darstellung 13

2.4.2 Brandschutzbestimmungen 13

3 Erstellung des Programms **14**

3.1 Vorraussetzungen 14

3.2 Nutzung des Grundriss 16

3.3 Erstellung des Kabelkanalnetzwerks 16

3.4 Optimale Wege berechnen 17

3.5 Erläuterung der Konfigurationsdatei 21

3.6 Erläuterung des Speicherformats 22

3.7 Verwendete Frameworks 23

3.8 Die Programmstruktur 23

4 Benutzerhandbuch **24**

4.1 Start des Programms 24

4.2 Knoten einfügen 25

4.3 Kanten einfügen 25

4.4 Knoten löschen 26

4.5 PDF-Datei einlesen 26

4.6 NDML einlesen und Geräte zuweisen 26

4.7 Geräte und Beschriftungen einfügen 27

4.8 Berechnung der kürzesten Wege 27

4.9 Ausdrucken 27
4.10 Das Menü 28
4.11 Konfiguration von CaNDeL 28

5 Resümee **28**

Anhang **I**

Glossar I
Literatur II
Eigenständigkeitserklärung V

Tabellenverzeichnis

1 Verschiedene Kabeltypen und deren Maximallängen 11
2 Farbzuordnung in der properties_version_1.dtd 29

Abbildungsverzeichnis

1 Kabelkanäle in verschiedenen Ausführungen 7
2 Richtungsänderung eines Kabelkanals 7
3 Aufsplittung bzw. Zusammenführung eines Kabelkanals 7
4 Zuerst ermittelte kürzeste Wege A ⇒ B und C ⇒ B 10
5 Ermittelte kürzeste Wege C ⇒ B und A ⇒ B 10
6 Schaltzeichen für eine Steckdose nach DIN 40900 14
7 Ausgabe der berechneten Länge und der Maximallänge 15
8 Beispielnetzwerk 17
9 Beispielnetzwerk (Abbildung 8 auf Seite 17) als Graph 18
10 Zwei unterschiedliche gleiche Graphen 18
11 Einfacher Graph 19
12 Paketstruktur des Programms VI
13 Inhalt des Ordners CaNDeL VII
14 Knoten zeichnen auswählen VII
15 Knoten gezeichnet VII
16 Kanten einfügen auswählen VII
17 Neue Kante hinzugefügt VIII
18 Verschieben eines Knoten VIII
19 Mehrere Knoten selektieren VIII
20 PDF einlesen wählen VIII
21 Eingabefenster für den Maßstab IX
22 NDML einlesen wählen IX
23 Kapazität einer Kante festlegen IX
24 Knoten ein Gerät zuweisen X
25 Berechnung der kürzesten Wege starten X
26 Das Ergebnis einer Berechnung XI
27 Zuvor berechnetes Ergebnis nochmals anzeigen XI
28 Graph ausdrucken XI

Liste der Algorithmen

1 Bellman-Ford-Algorithmus 8
2 Algorithmus von Dijkstra 9
3 Algorithmus zur Berechnung der optimalen Kabelführung 20

Listings

1 ndml_version_1.1.dtd 12
2 NDML-Beschreibung eines Netzwerks mit zwei Rechnern 13
3 properties_version_1.dtd 21
4 save_version_1.dtd 22
5 ISolver.java 24
6 IReturnData.java 25
7 ITriple.java 26

1 Einführung

CANDY ist ein Projekt des Lehrstuhls für Rechnernetze unter der Leitung von Prof. Dr. habil. A. Schill an der Fakultät Informatik der TU Dresden zur Erstellung einer Software im Umfeld der Netzwerkprojektierung. Ziel des Projektes ist es, eine Arbeitsumgebung zu schaffen, mit der man die Vernetzung in Neubauten, von der Topologie über die Kosten bis hin zur Leitungsführung, ermitteln kann. Weitere Informationen siehe [18].

Im Rahmen dieser Belegarbeit sollte die CANDY-Umgebung um eine weitere Funktionalität ergänzt werden. Ziel war die Erstellung eines Programms, mit dem man ein Netzwerk aus Kabelkanälen erstellen kann und dafür die optimale Kabelführung ermittelt.

1.1 Erläuterung der Aufgabenstellung

Die erste Aufgabe bestand demnach darin, sich mit vorhandenen Algorithmen zu beschäftigen und diese zu analysieren und zu testen, ob sie für diese Aufgabenstellung geeignet sind. Zu diesem Zweck habe ich mich näher mit dem Bellman-Ford-Algorithmus und dem Dijkstra-Algorithmus auseinander gesetzt. Des Weiteren galt es sich mit den gesetzlichen Bestimmungen und den gültigen DIN-Normen zu beschäftigen, um diese bei der Implementierung zu berücksichtigen. Zusätzlich musste noch eine passende Repräsentation der Realität, sprich dem Grundriss mit seinen Kabelkanälen, gefunden werden, um diese möglichst einfach und intuitiv zu gestalten, so dass sich der Benutzer des Programms schnell zurecht finden kann, ohne vorher lange die Anleitung studiert zu haben. Letztendlich musste das entstandene Programm natürlich auch noch ausgiebig getestet werden.

2 Vorarbeit

Bevor das Programm realisiert werden konnte, mussten einige Nachforschungen angestellt werden. So wird ein Algorithmus benötigt, der den optimalen Kabelverlauf berechnet. Dazu galt es, eine geeignete Repräsentation des Sachverhalts, ein Grundriss mit eingezeichneten Kabelkanälen, zu finden, um überhaupt einen Algorithmus anwenden zu können. Schließlich mussten die Algorithmen selbst auf ihre Tauglichkeit geprüft werden. Des Weiteren soll das Programm auf den Ergebnissen der vorherigen CANDY-Programmstufen aufbauen. Arbeitsgrundlage für diese ist eine NDML-Datei, eine CANDY-eigene XML-Spezifikation, die alle wichtigen Daten des Netzwerks enthält. Um diese verwenden zu können, musste diese analysiert und die für das zu erstellende Programm wichtigen Spezifikationen ermittelt werden.

Bei der Erbauung von Gebäuden müssen viele Gesetze und Richtlinien berücksichtigt und beachtet werden. Das trifft auch für die Verkabelung zu. Deswegen galt es entsprechende Nachforschungen anzustellen, um gegebenenfalls wichtige Sachverhalte bei der Implementierung zu berücksichtigen. Wichtig ist auch die graphische Darstellung, damit sich der Designer möglichst schnell mit dem Programm zurechtfinden kann und auch der Elektroinstallateur, der im Anschluss die einzelnen Netzwerkkabel verlegen soll, sofort erblicken kann, wo er welches Kabel verlegen soll.

2.1 Analyse der Realität

Um ein Programm zu realisieren, muss man zuerst die reale Welt studieren, um diese verstehen zu können. Erst mit diesem Verständnis kann ein anwendergerechtes Programm geschrieben werden, welches das gegebene Problem zufrieden stellend löst. Wichtig beim Hausbau ist der Grundriss. Dieser gibt vor, wo welche Wand verläuft, wie hoch die Decken sind und vieles mehr.

Aber auch die Kabelkanäle selbst bedürfen einer genaueren Betrachtung, da diese die eigentliche Basis für die Berechnung der optimalen Wege bilden.

2.1.1 Der Grundriss

Im Allgemeinen ist es so, dass der Verlauf der Kabelkanäle noch nicht im Grundriss enthalten ist. Diese werden erst nachträglich von der für das Netzwerk zuständigen Person eingetragen, wenn das Netzwerk fertig konzipiert ist. Und dies stellt genau die Aufgabe von CANDY dar. Es muss also ein Weg gefunden werden, wie der Designer optimal bei seiner Aufgabe unterstützt werden kann. Dies kann geschehen, indem er vor der Nutzung dieses Programms von Hand die Kanäle in den Grundriss einzeichnet und das Programm diese Zeichnung auswertet. Eine andere Möglichkeit ist, dass der Designer die Kanäle am PC einzeichnen kann. Natürlich muss es dann möglich sein, dass skizzierte Kabelkanalnetzwerk auszudrucken, um dem Bauleiter die nötigen Instruktionen erteilen zu können, wie die Kanäle verlegt werden sollen.

Die erste Variante hat den Vorteil, dass das einzeichnen der Kanäle schnell erledigt werden kann und kein PC benötigt wird. Allerdings ist es hinterher sehr schwer und fehleranfällig, den Grundriss einzulesen und die Kanäle maschinell zu identifizieren. Außerdem muss bei einer Änderung des Netzwerks diese im Grundriss vorgenommen und anschließend erneut eingelesen werden.

Die zweite Variante hat den Vorteil, dass das Netzwerk direkt am PC erstellt werden kann und Änderungen leicht möglich sind. Zudem entfällt die Fehlerquelle, das beim Einlesen des Grundrisses Kanten nicht erkannt bzw. „falsche“ Kanten ermittelt werden. Nachteilig ist, dass eine Druckfunktion und/oder Speicherfunktion realisiert werden muss, um auch später noch auf das erstellte Netzwerk zugreifen zu können.

2.1.2 Der Kabelkanal

Kabelkanäle gibt es in den verschiedensten Ausführungen, wie das Bild auf Seite 7 zeigt. Es gibt offene Kanäle, geschlossene Kanäle, Unterputz-Kanäle, Aufputz-Kanäle, eckige Kanäle, runde Kanäle, Kanäle aus Kunststoff, Kanäle aus Metall, nur um ein paar Beispiele zu nennen. Im Hausgebrauch werden sie meist genutzt, um bspw. die Zuleitung für den Fernseher zu verstecken oder den Kabelsalat hinter dem PC etwas zu ordnen. Man kann sie aber auch gebäudeweit einsetzen und alle Leitungen des Gebäudes in Kanälen verlegen. Dies hat den Vorteil, dass man genau weiß, wo die Leitungen verlaufen und diese zusätzlich geschützt sind. Des Weiteren kommt man im Bedarfsfall relativ schnell an sie heran, wenn zum Beispiel ein Kabel defekt ist oder die Netzwerktechnik gewechselt wird und alle Netzwerkkabel ausgetauscht werden müssen. Auch ist es bei ausreichender Größe der Kanäle kein Problem, nachträglich noch weitere Kabel zu verlegen, wenn dies nötig wird.

Aufgabe des Designers ist es, sich Gedanken um den Verlauf der Kanäle zu machen und diese richtig zu dimensionieren, um die Wartungskosten des Kabelnetzwerks so gering wie möglich zu halten. Diese Kosten können zum Beispiel die Behebung von

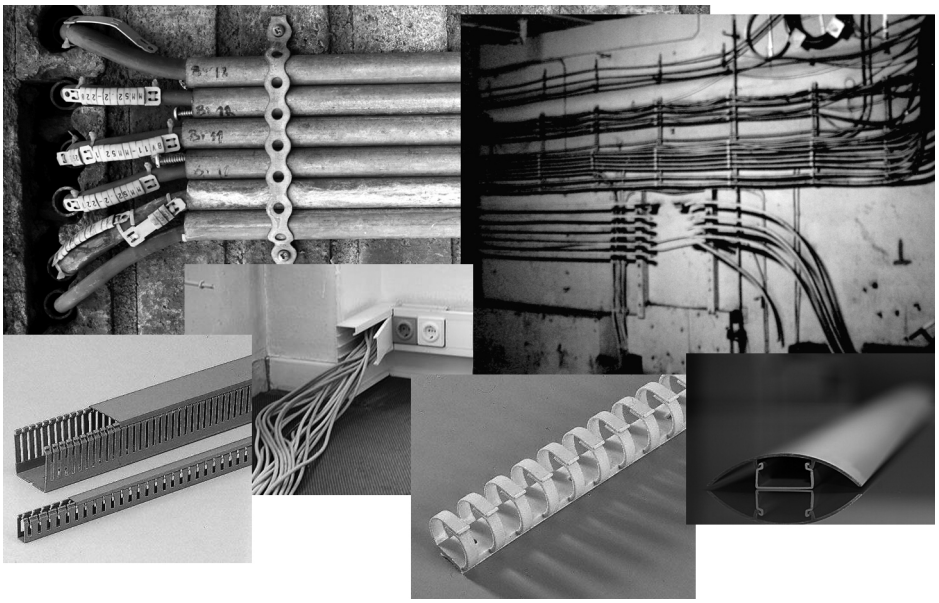

Abbildung 1: Kabelkanäle in verschiedenen Ausführungen

Fehlern im Leitungssystem sein, der Austausch von Leitungen oder das Verlegen zusätzlicher Leitungen.

Selbstverständlich können sich die Kanäle auch aufsplitten bzw. zusammengeführt (Abbildung 3 auf Seite 7) werden und sich den baulichen Gegebenheiten des Gebäudes anpassen (Abbildung 2 auf Seite 7). Auch muss berücksichtigt werden, dass die Kanäle nicht nur waagerecht verbaut werden, um eine Etage zu versorgen, sondern auch senkrecht, um bspw. mehrere Etagen zu verbinden oder um den WLAN-Router unter der Decke anschließen zu können.

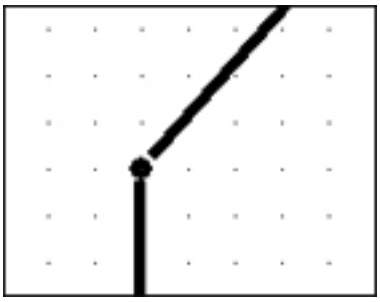

Abbildung 2: Richtungsänderung

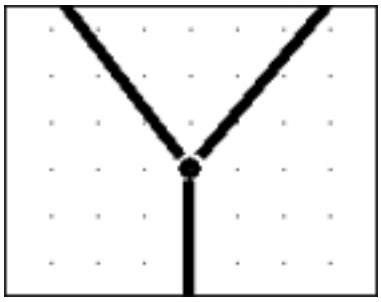

Abbildung 3: Aufsplittung

2.2 Analyse vorhandener Algorithmen

2.2.1 Bellman-Ford-Algorithmus

Während meiner Vorbereitung habe ich mich näher mit dem Bellman-Ford-Algorithmus beschäftigt. Dieser gehört der Gruppe der dynamischen Algorithmen an. Das Konzept der dynamischen Algorithmen bzw. Programmierung ist vornehmlich

von Herrn Bellman, einem amerikanischen Mathematiker, in den 1940er Jahren entwickelt worden. Es geht davon aus, dass sich einige Optimierungsprobleme in kleinere Teilprobleme aufteilen lassen. Die Optimallösung dieser Teilprobleme ergibt die Optimallösung des Gesamtproblems (Optimalitätsprinzip von Bellmann [7]). Ein Beispiel hierfür ist die Berechnung der kürzesten Wege in einem Graphen. Es gibt einen kürzesten Weg A in einem Graphen zwischen den Knoten X und Y. Zusätzlich führt der Weg A auch über die Knoten U und V. Nach dem Optimalitätsprinzip muss dann auch der Weg zwischen U und V optimal (d.h. so kurz wie möglich) sein. Wenn das nicht der Fall wäre, dann könnte man den Weg A verkürzen, in dem man einen kürzeren Weg zwischen U und V wählt. Damit steht es aber im Widerspruch zu der Annahme, dass A der kürzeste Weg war.

Der von Bellman und Lester Ford entwickelte Algorithmus ähnelt sehr dem Floyed-Warshall-Algorithmus. Beide dienen der Berechnung der kürzesten Wege in einem Graphen. Der Bellman-Ford-Algorithmus in Pseudocode ist auf Seite 8 zu finden.

```
für alle v ∈ V tue
    Distanz(v) = ∞
    Vorgänger(v) = kein
Distanz(s) = 0

wiederhole n-1 mal
    für jedes (u, v) ∈ E tue
        wenn Distanz(u) + Gewicht(u, v) < Distanz(v) dann
            Distanz(v) = Distanz(u) + Gewicht(u, v)
            Vorgänger(v) = u

für jedes (u, v) ∈ E tue
    wenn Distanz(u) + Gewicht(u, v) < Distanz(v) dann
        STOP mit Ausgabe "Zyklus negativer Länge gefunden"

Ausgabe : Distanz
```

Algorithmus 1 : Bellman-Ford-Algorithmus [2]

Die Menge `V` in dem skizzierten Bellman-Ford-Algorithmus enthält die Knoten des Graphen, `E` ist die Menge der Kanten, die auch ungerichtet sein können, `Gewicht` ist die Gewichtsfunktion für die Kanten. Die Gewichtsfunktion kann auch negative Werte liefern. `s` ist der Startknoten, `U` ist die Menge der noch zu bearbeitenden Knoten und `n` die Anzahl der Knoten (Kardinalität von `V`).

Die Komplexität des Algorithmus liegt bei $\mathcal{O}(|V| \cdot |E|)$, wobei $|V|$ die Anzahl der Knoten und $|E|$ die Anzahl der Kanten im Graphen ist.

Es sei auch noch auf die Arbeit [13] verwiesen, in der sich auch schon näher mit der dynamischen Optimierung nach Bellman befasst worden ist und Ansätze geschildert werden, die zu einer Lösung des Problems führen können.

2.2.2 Dijkstra-Algorithmus

Der von Edsger Wybe Dijkstra entwickelte Algorithmus gehört zu der Klasse der Greedy-Algorithmen (zu deutsch: gierige Algorithmen). Diese zeichnen sich dadurch aus, dass sie immer den Folgezustand wählen, der zum jetzigen Zeitpunkt den größten Gewinn bzw. das beste Ergebnis verspricht (lokales Optimum). Hieraus kann sich auch ein globales Optimum ergeben. Oft wird diese Art von Algorithmen an-

gewendet, um NP-vollständige Probleme lösen zu können. Diese Berechnung führt dann zwar meist nicht zu einem optimalen Ergebnis, aber zu einer sehr guten Annäherung. Im Falle des Dijkstra-Algorithmus erhält man ein optimales Ergebnis.

Der Algorithmus dient dazu, in einem bewerteten ungerichteten bzw. gerichteten Graphen alle kürzesten Wege von einem bestimmten Startknoten aus zu berechnen. Allerdings darf die Bewertung der Kanten im Gegensatz zum Bellman-Ford-Algorithmus nicht negativ sein. Der Dijkstra-Algorithmus in Pseudocode ist auf Seite 9 zu finden. Dabei ist die Menge V ist die Menge der Knoten, E die Menge

```
für jedes v ∈ V tue
    Distanz(v) = ∞
    Vorgänger(v) = kein
Distanz(s) = 0
Vorgänger(s) = 'begin'
U = V
M = ∅
für jedes (s, v) ∈ E mit v ∈ U tue
    Distanz(v) = Kosten(s, v)
    M = M ∪ v
solange M ≠ ∅ tue
    wähle u aus M mit Distanz(u) minimal
    M = M \ u
    U = U \ u
    für jedes (u, v) ∈ E mit v ∈ U tue
        M = M ∪ v wenn Distanz(u) + Kosten(u, v) < Distanz(v) dann
            Distanz(v) = Distanz(u) + Kosten(u, v)
            Vorgänger(v) = u
Ausgabe : Distanz und Vorgänger
```

Algorithmus 2 : Algorithmus von Dijkstra [1]

der Kanten, $Kosten$ die Gewichtsfunktion für die Kanten, s der Startknoten und U die Menge der noch zu bearbeitenden Knoten.

Nach dem Durchlauf des Algorithmus muss allerdings noch ein zweiter Algorithmus gestartet werden, der aus dem Vorgänger den minimalen Spannbaum (entspricht den kürzesten Wegen) ermittelt. Dies stellt kein großes Problem mehr dar und soll an dieser Stelle vernachlässigt werden.

Die Komplexität des Dijkstra-Algorithmus liegt bei $\mathcal{O}(n^2)$, wobei n die Anzahl der Knoten ist. Durch die Verwendung eines Fibonacci-Heap anstelle einer Knotenliste beträgt die Laufzeit nur noch $\mathcal{O}(m + n \cdot \log(n))$, wobei m die Anzahl der Kanten ist.

2.2.3 Resümee für die Implementierung

Die beiden betrachteten Algorithmen sind für die Lösung des vorliegenden Problems nur bedingt geeignet. Beide Verfahren nutzen als Kriterium für die Wahl des kürzesten Weges die Gewichtsfunktion, also die Kosten, die es verursacht, wenn man eine Kante nutzt. Diese Kosten kann man mit den Längen der Kanten gleichsetzen. Nach dem Durchlauf der Algorithmen sind die kostengünstigsten Wege in einem Graphen ermittelt worden, also in diesem Fall die Wege mit der geringsten Länge.

Dies ist aber für das gegebene Problem nicht ausreichend.

Ein wichtiges Kriterium bei der Wegewahl ist auch noch die Kapazität der einzelnen Kanten (siehe dazu „Analyse der Realität“ auf Seite 6). Wenn zu viele kürzeste Wege über eine Kante führen, kann es dazu kommen, dass die Kapazität überschritten wird. In diesem Fall muss nach einem alternativen Weg gesucht werden. Dieser „neue Weg“ wird zwar länger als der „alte Weg“ sein, ansonsten wäre er ja schon beim ersten Durchlauf gefunden worden, aber ohne diese Nachrechnung gäbe es gar keinen möglichen Weg. Zu beachten ist auch, dass die Nachrechnung nicht erst ab dem Knoten gestartet werden kann, von dem die volle Kante abgeht, da das Ziel lauten muss, den zweitkürzesten Weg im Graphen zu finden. Und dies muss nicht gegeben sein, wenn man den Algorithmus erneut von diesem Knoten starten lässt, und nicht von dem Ursprungsknoten der Berechnung, wie man anhand der Abbildung 4 auf Seite 10 leicht sehen kann. Dies stellt die gefundenen Wege von A ⇒ B und C ⇒ B dar, wobei die Kanten alle eine Kapazität von eins haben. Der Weg A ⇒ B ist zuerst ermittelt worden. Die durchgezogene Linie kennzeichnet die Wegführung nach einem Neustart des Algorithmus an Knoten D und die gestrichelte Linie die Wegführung nach einem Neustart am Ausgangsknoten C.

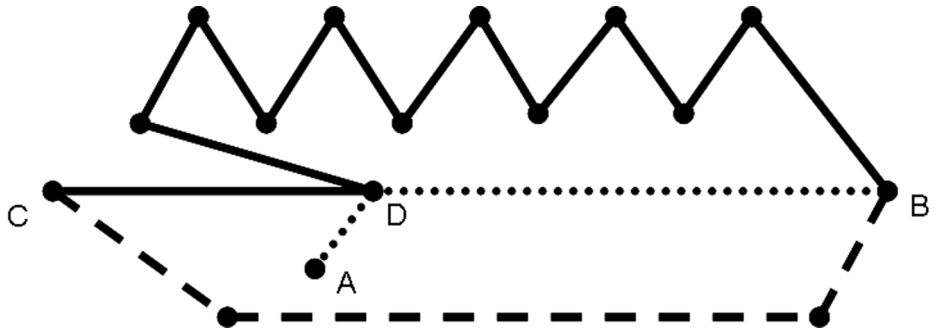

Abbildung 4: Zuerst ermittelte kürzeste Wege A ⇒ B und C ⇒ B

An dieser Abbildung lässt sich auch leicht erkennen, dass auch die Reihenfolge, in der die Wege berechnet werden, unter der Berücksichtigung, dass die Kanten nur eine begrenzte Kapazität haben, einen großen Einfluss auf die ermittelten kürzesten Wege hat. Wäre zuerst der Weg C ⇒ B berechnet worden, und anschließend der Weg A ⇒ B, hätte sich ein ganz anderes Ergebnis ergeben, wie man anhand der Abbildung 5 auf Seite 10 sehen kann. Die Kapazität der Kanäle ist wieder eins. Der

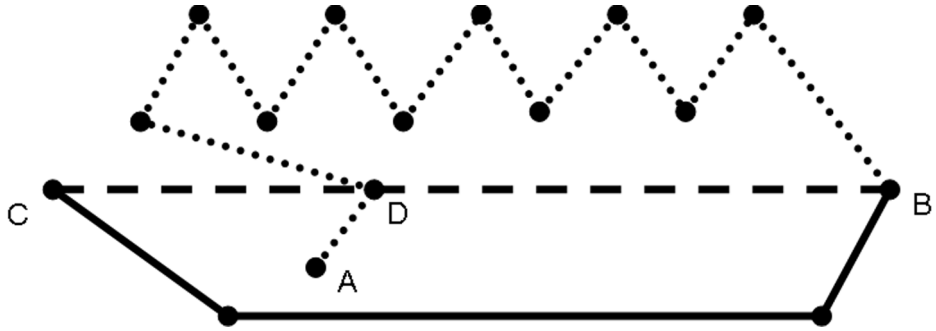

Abbildung 5: Ermittelte kürzeste Wege C ⇒ B und A ⇒ B

Weg C ⇒ B ist zwar jetzt etwas kürzer gegenüber dem gestrichelten Weg aus der Abbildung 4, dafür ist der Weg A ⇒ B wesentlich länger geworden.

Des Weiteren beachten die Algorithmen nicht, dass die einzelnen kürzesten Wege bestimmte Längen nicht überschreiten dürfen, da sonst die maximalen Längen der verschiedenen Netzwerkkabeltypen überschritten werden könnten. Wie man anhand

Kabeltyp	**Standard**	**Länge**[1]
RG58 Kabel	10Base2	185 m
RG8 Kabel	10Base5	500 m
CAT-3 Kabel	10Base-T	100 m
CAT-5 Kabel	10Base-T	100 m
	100Base-T	100 m
Multimode-Glasfaser	1000Base-SX	550 m
	10GBase-SR	82 m
	10GBase-LX4	300 m
Singlemode-Glasfaser	1000Base-SX	2.000 m
	1000Base-LX	10.000 m
	10GBase-LX4	10.000 m
	10GBase-ER	40.000 m
Monomodeglasfaser	1000Base-LX	10.000 m
STP	1000Base-CX	25 m
Doppelt-twinaxiale Kupferkabel	10GBase-T	100 m

Tabelle 1: Verschiedene Kabeltypen und deren Maximallängen [4], [5]

der Tabelle 1 auf Seite 11 sehen kann, ist es aber gar nicht so leicht, eine Aussage zu treffen, wie lang ein Kabel sein darf. So kann ein und dasselbe Kabel bei identischer Übertragungsrate unterschiedlich lang sein, nur bedingt durch die Verwendung einer anderen Übertragungstechnik. Hat man dieses Problem gelöst, so könnte man den Algorithmus terminieren, sobald dieser während der Berechnung feststellt, dass der bis jetzt berechnete Weg bereits länger ist, als es die Maximallänge des Kabels zulässt.

Der zu implementierende Algorithmus muss demnach zwei Kriterien bei der Wegewahl berücksichtigen. Zum einen die Länge der Kanten und zum anderen die Kapazität der Kanten. Des Weiteren ist die Reihenfolge, in der die kürzesten Wege berechnet werden, wichtig.

2.3 Analyse der NDML

Um Berechnen zu können, wie die Kabel verlegt werden soll, werden die Daten der Netzwerkinfrastruktur benötigt. Diese sollten laut Aufgabenstellung der NDML-Beschreibung entnommen werden, die in den vorangegangenen Programmstufen von CANDY erstellt worden ist. Hierbei handelt es sich um ein CANDY-eigenes XML-Format, welches durch eine DTD definiert wird. Zur Zeit der Belegerstellung befand diese sich in der Version 1.1. Nach Fertigstellung lag eine Weiterentwicklung zu der Version 2.0 vor und Überlegungen für eine Version 3.0 (siehe [14]). Da es sich aber bei den Neuerungen eher um strukturelle als um inhaltliche Veränderungen geht und somit der Informationsgehalt in etwa gleich geblieben ist, basiert diese

[1] Dient lediglich als Richtwert. Hängt stark von den eingesetzten Technologien, wie Hubs oder Switches ab. Bezeichnet die Gesamtkabellänge oder die Länge des Kabel zwischen zwei Stationen.

Arbeit dennoch auf der NDML-Version 1.1. Dies führt auch dazu, das eigene XML-Definitionen durch eine DTD beschrieben werden und nicht, wie bei den neueren NDML-Varianten, durch ein XML Schema. Allerdings sind die DTD leicht in das XML-Schema überführbar, so dass an dieser Stelle kein großer Nachteil entsteht. Die für dieses Programm wichtigsten Definitionen der NDML Version 1.1 kann man dem Listing 1 auf Seite 12 entnehmen. Es werden Geräte (*device*) definiert. Diese

```
<!ELEMENT devices (device*)>

<!ELEMENT device (l_agent*)>
<!ATTLIST device
      id ID #REQUIRED
      ...
>

<!ELEMENT nics (nic*)>

<!ELEMENT nic (l_nicType, technology)>
<!ATTLIST nic
      id ID #REQUIRED
      home IDREF #REQUIRED
      ...
>

<!ELEMENT media (medium*)>

<!ELEMENT medium (connectedNic+)>
<!ATTLIST medium
      id ID #REQUIRED
      length CDATA #IMPLIED
      ...
>

<!ELEMENT connectedNic EMPTY>
<!ATTLIST connectedNic
      id IDREF #REQUIRED
>
```

Listing 1: ndml_version_1.1.dtd

stellen die einzelnen Rechner, Switches usw. des Netzwerks dar. Neben anderen Eigenschaften müssen die Geräte jeweils eine feste ID haben. Ihnen lassen sich Schnittstellen (*nic*) zuweisen (*home*). Diese stellen die einzelnen Buchsen dar, in die man die Netzwerkkabel hineinstecken kann. Untereinander verbunden werden die Schnittstellen über Leitungen (*medium*), wobei unter Leitungen nicht nur Kabel, sondern auch kabellose Medien, wie Satellitenübertragung und WLAN, zu verstehen sind.

Eine NDML-Beschreibung für ein Netzwerk mit zwei Rechnern, die direkt miteinander verbunden sind, könnte so aussehen, wie es auf Seite 13 dargestellt ist.

Für das Programm muss also eine geeignete Struktur gefunden werden, um den Sachverhalt der NDML-Datei auf den schon mehrfach erwähnten Graphen zu übertragen. Es muss ermittelt werden, welche Geräte untereinander verbunden werden sollen. Dazu ist es notwendig, die Liste der Leitungen durchzugehen, um festzustellen, welche Schnittstellen miteinander verbunden werden sollen. Um die Usability des Programms zu erhöhen, wäre es schön, wenn der Nutzer zusätzlich noch die Information bekommt, welche Schnittstelle zu welchem Gerät gehört, da ihm so die Zuordnung wesentlich erleichtert wird.

```
 \dontprintsemicolon
...
<devices>
     <device id="de1" name="Rechner 1"
         description="Dies ist der Rechner 1" />
     <device id="de2" name="Rechner 2"
         description="Dies ist der Rechner 2" />
</devices>

<nics>
     <nic id="ni1" description="" home="de1" ...>
          <l_nicType l_wireless="OFF" l_satelliteOriented="OFF"
              l_wired="ON" />
          <technology>100 Base T</technology>
     </nic>
     <nic id="ni2" description="" home="de2" ...>
          <l_nicType l_wireless="OFF" l_satelliteOriented="OFF"
              l_wired="ON" />
          <technology>100 Base T</technology>
     </nic>
</nics>

<media>
     <medium id="me1" name="UTP CAT 5" length="100.0">
          <connectedNic id="ni1" />
          <connectedNic id="ni2" />
     </medium>
</media>
...
```

Listing 2: NDML-Beschreibung eines Netzwerks mit zwei Rechnern

2.4 Gesetzliche Bestimmungen und Normen

Zu diesem Bereich gehört die graphische Darstellung von Netzwerkkabeln in einem Grundriss, sowie die Verlegepläne. Aber auch etwaige Brandschutzbestimmungen, die zum Beispiel die Art der Kabelführung in öffentlichen Gebäuden regeln, müssen beachtet werden.

2.4.1 Graphische Darstellung

Bei der Recherche musste ich feststellen, dass es anscheinend noch kein Symbol für ein Netzwerkkabel in einem Grundriss nach derzeit gültiger DIN gibt. Stattdessen wird das Schaltzeichen für Steckdosen verwendet und mit EDV beschriftet (Abbildung 6 auf Seite 14) [20]. Ebenso scheint es auch keine einheitliche Regelung zu geben, wie man den Verlauf von Kabeln in einen Grundriss einzuzeichnen hat.

2.4.2 Brandschutzbestimmungen

Kabel, auch Netzwerkkabel, sollten und dürfen nicht beliebig verlegt werden. Dies macht unter anderem das Auffinden von Kabeln in verputzten Wänden viel einfacher. So stehen zum Beispiel viele Hobby-Handwerker oft vor dem Problem, ob sie an der gewünschten Stelle ein Loch bohren können oder nicht, weil sich in der Näher eine Steckdose befindet. Wenn sich der Elektriker an die Richtlinien gehalten hat, dann lässt sich das Problem sehr leicht aus der Welt schaffen. Die Kabel sollten von der Stelle ihres Austretens senkrecht nach unten führen und erst knapp über dem Fußboden horizontal weiter geführt werden.

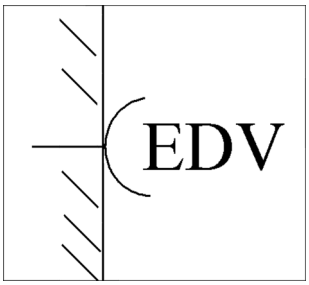

Abbildung 6: Schaltzeichen für eine Steckdose nach DIN 40900 mit zusätzlicher Beschriftung

Neben dieser „Vereinfachung“ gibt es noch viele weitere Bestimmungen, bspw. um die Brandgefahr im Kurzschlussfall so gering wie möglich zu halten. Erwähnt seien hier unter anderem das Bauproduktgesetz der EG vom 10.08.1992, das EG Grundlagendokument 2: Brandschutz 02.94, die Musterbauverordnungen, die Landesbauordnungen, die Sonderbauten-Richtlinien und etwaige Genehmigungen im Einzelfall [21].

3 Erstellung des Programms

Aus den vorausgegangenen Vorüberlegungen muss eine Vorgehensweise abgeleitet werden, wie das Programm entwickelt werden soll. Zudem müssen die Ziele definiert und mögliche Einschränkungen vorgenommen werden, um das Programm realisieren zu können.

3.1 Vorraussetzungen

Unter der Rubrik „Gesetzliche Bestimmungen und Normen“ auf Seite 13 wurde bereits die gesetzliche Sachlage etwas hinterleuchtet. Die richtige Kabelverlegung wird von vielen Einflussfaktoren bestimmt. Es kommt auf die Beschaffenheit der einzelnen Räume an, die dort verwendeten Baumaterialien und die Verläufe der übrigen Kabel und Leitungen. Des Weiteren üben auch noch regionale Bestimmungen Einfluss auf das Bauvorhaben aus. Da diese Daten aber nicht in der NDML-Beschreibung vorliegen und der Nutzer des Programms mit der Eingabe dieser Daten nicht überfordert werden soll, wird keinerlei Rücksicht bei der Berechnung der optimalen Wege auf diese Sachverhalte genommen. Es wird davon ausgegangen, dass der Designer selbst einen guten Kenntnisstand hat und diesen nutzt, um eventuelle Fehler bei der Verlegung der Kabelkanäle zu vermeiden.

Ebenso wird davon ausgegangen, dass die verschiedenen Kabeltypen alle den gleichen Durchmesser besitzen. Somit kann man die Kapazität eines Kanals mit der Anzahl an Kabeln, die der Kanal aufnehmen kann, gleichsetzen. Angenommen, ein Kanal hat die Kapazität fünf, so lassen sich in diesem Kanal fünf Kabel unterbringen. Dies erleichtert zum Einen die Berechnung der optimalen Wege, aber auch für den Anwender bringt es Vorteile. Die NDML-Beschreibung des Netzwerks bietet bis jetzt keine Möglichkeit an, den Durchmesser eines Kabels zu spezifizieren. So müsste der Nutzer für jede Kabelart dazu aufgefordert werden, deren Durchmesser anzugeben. Eine zweite Erleichterung für den Nutzer ergibt sich daraus, dass jeder

Kanal eine Standardkapazität von fünf hat. Nur wenn der Kanal eine abweichende Größe hat, muss diese individuell angepasst werden.

Ein weiteres Problem der NDML-Beschreibung ist, dass die Schnittstellen zwar ein Attribut *technology* (siehe „Analyse der NDML“ auf Seite 11) besitzt, allerdings ist dieses vom Typ PCDATA und kann somit alle möglichen Zeichensequenzen beinhalten. Um das Feld für die Auswertung nutzen zu können, müsste es eine Definition geben, welche Zeichenketten zulässig sind und was diese genau bezeichnen. Allerdings würde diese Information alleine nicht ausreichen, um zu ermitteln, wie Lang ein Kabel maximal sein darf. Wie sich im „Resümee für die Implementierung“ auf Seite 9 gezeigt hat, übt auch die Wahl des Kabels eine große Rolle auf die maximale Länge aus. Um diesen Problem aus dem Weg zu gehen, wird stattdessen das Attribut *length* des *mediums* ausgewertet. Es wird davon ausgegangen, dass dies die maximale Länge des Kabels zwischen zwei Geräten ist. Da das Attribut nur optional ist, wird bei dem Fehlen dieser Angabe von einer maximalen Länge von 100 Metern ausgegangen.

Da das Programm nur die optimale Verlegung von Netzwerkabeln berechnen soll, werden nur Leitungen mit in die Berechnung einbezogen, bei denen das Attribut *l_nicType* in *nic* in der NDML-Bescheibung wie folgt gesetzt ist:

- `l_wireless=‘‘OFF’’`
- `l_satelliteOriented=‘‘OFF’’`
- `l_wired=‘‘ON’’`

Zusätzlich müssen die einzelnen Leitungen mindestens zwei *connectedNic* besitzen. Da im Allgemeinen ein Kabel nur zwei Enden hat, werden bei mehr als zwei eingetragenen *connectedNics* nur die ersten beiden Einträge ausgewertet.

Der einfachheit halber arbeitet das Programm nur im zwei-dimensionalen Raum. Das heißt, wenn eine Berechnung für eine waagerechte Verkabelung gestartet wird, so können senkrechte Kabelkanäle nicht berücksichtigt werden und umgekehrt. Um dies dennoch bei der Planung berücksichtigen zu können, berechnet das Programm nicht nur die Länge des Kanals und gibt diese aus, sondern es wird auch die maximale Länge des Kabels ausgegeben (Abbildung 7 auf Seite 15), welches in diesen Kanal gelegt werden soll. So kann man einfach eine Abschätzung machen, wie viel Reserve das Kabel noch bietet und anhand dessen abschätzen, ob eine senkrechte Weiterführung noch möglich ist.

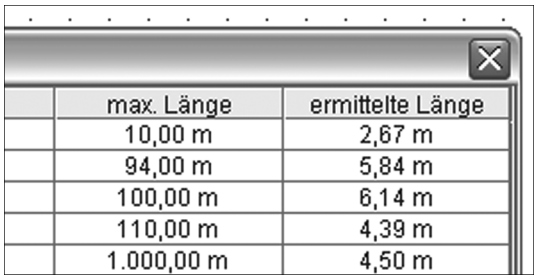

Abbildung 7: Ausgabe der berechneten Länge und der Maximallänge

Unter „Erstellung des Kabelkanalnetzwerks“ auf Seite 16 wird erläutert, dass die Länge der Kanäle aus den Längen der Kanten ermittelt werden. Damit die Umrech-

nung der Längen der graphischen Darstellung in metrische Maße korrekt durchgeführt wird, muss der Monitor mit einer Auflösung von 72 DPI arbeiten. Im Normalfall stellt dies keine Einschränkung der Nutzung des Programms dar, da die meisten handelsüblichen Monitore in dieser Auflösung arbeiten.

Das Programm nutzt einige Features, wie erweiterte for-Schleifen und generische Datentypen, die erst mit der neuen Version 1.5 von Java eingeführt worden sind. Um das Programm ausführen zu können, wird deshalb Java in der Version 1.5 benötigt.

3.2 Nutzung des Grundriss

Aus den Betrachtungen unter „Der Grundriss“ auf Seite 6 liegt es nahe, das Kabelkanalnetzwerk digital und nicht per Hand zu erstellen. So kann das Programm relativ einfach gehalten werden und es entfällt das Durchsuchen des gescannten Grundriss nach den Kanälen. Um dem Designer das Erstellen des Kanalnetzwerks zu erleichtern, bietet es sich an, den Grundriss in den Hintergrund der Zeichenfläche zu legen. Für diese Zwecke eignet sich das PDF-Format sehr gut. Es ist genauso wie das Programm selbst, welches in Java realisiert ist, plattform- und systemunabhängig. Auch ist die Erzeugung einer PDF-Datei relativ einfach. Viele Programme bieten von sich aus schon einen PDF-Export an. Ist diese Möglichkeit nicht gegeben, so kann man eine PDF-Datei erzeugen, indem die Datei auf einem virtuellen Drucker ausgedruckt wird, der dann aus den Druckdaten eine PDF-Datei erzeugt.

3.3 Erstellung des Kabelkanalnetzwerks

Aus den unter „Analyse der Realität“ auf Seite 6 genannten Gründen und der Entscheidung im vorangegangenen Abschnitt muss das Programm eine Möglichkeit bieten, dass der Nutzer ein Kabelkanalnetzwerk erstellen kann. Ein solches Netzwerk besteht aus im Allgemeinen mehreren Kanälen, die untereinander verbunden sind. Dabei kann es auch zu Abzweigungen, Kreuzungen und Richtungsänderungen kommen. An den Enden der Kanäle treten die Kabel aus diesen aus und werden mit den entsprechenden Geräten verbunden, so wie es in der Abbildung 8 auf Seite 17 dargestellt ist. Dieser Sachverhalt lässt sich gut mit einem ungerichteten Graphen wiedergeben. Die einzelnen Kanten des Graphen stellen die Kanäle dar, die Ecken des Graphen, zukünftig Knoten genannt, können dazu genutzt werden, Kreuzungen abzubilden, Richtungsänderungen vorzunehmen oder Endpunkte zu markieren (Abbildung 9 auf Seite 18). Die Endpunkte können mit den Gerätebezeichnungen beschriftet werden, die dort aufgestellt werden sollen. Diese können der NDML-Beschreibung des Netzwerks entnommen werden (siehe „Analyse der NDML“ auf Seite 11).

Bei den Kanten müssen zwei Dinge beachtet werden. Zum einen muss man eine Möglichkeit haben, deren Länge angeben zu können, um berechnen zu können, wie lang die gesamte Wegstrecke ist. Zum Anderen muss man die Kapazität der einzelnen Kanäle angeben können, d.h. wie viele Kabel eine Kanal aufnehmen kann. Da der Designer eine PDF-Datei mit dem Grundriss einlesen kann und diese im Hintergrund der Zeichenfläche dargestellt wird, bietet es sich an, die Länge der Kanäle implizit aus der Länge der Kanten auszulesen. Dies hat die Vorteile, dass der Designer das Netzwerk maßstabsgetreu erstellen und an dem Grundriss ausrichten kann und nicht explizit für jede Kante deren Länge angegeben werden muss. Allerdings führt das zu einer neuen Definition der Gleichheit zweier Graphen. Im Allgemeinen sind der Graph 1 mit der Knotenmenge V1, der Kantenmenge E1 sowie der

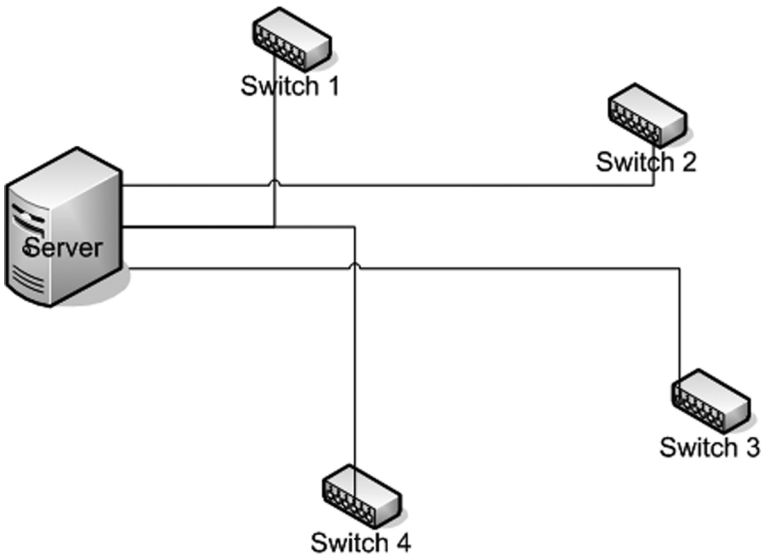

Abbildung 8: Beispielnetzwerk

Gewichtsfunktion G1 und der Graph 2 mit V2, E2 und G2 gleich, wenn gilt:

$$V1 \setminus V2 = \emptyset \quad \text{und} \quad E1 \setminus E2 = \emptyset \quad \text{und} \quad G1 \setminus G2 = \emptyset$$

, wobei sich die Namen der Knoten unterscheiden können. Das heißt, die Längen der Kanten und die Position der Knoten spielen keine Rolle. Diese Eigenschaft wird auch als Automorphismus bezeichnet.

Dies ist bei dem im Programm verwendeten Graphen nicht der Fall (Abbildung 10 auf Seite 18). Die Positionen der einzelnen Knoten spielen eine wichtige Rolle, da so die Kanten festgelegt werden und somit auch die Länge der Kanten. Die Kapazität der Kanten wird durch deren Beschriftung festgelegt. Entfällt diese, wird von einer Standardkapazität von fünf ausgegangen (siehe „Vorraussetzungen“ auf Seite 14).

3.4 Optimale Wege berechnen

Kernstück des Programms ist der Algorithmus zur Berechnung der kürzesten Wege in einem Graphen. Da das Programm erst einmal nur die technischen Möglichkeiten ausloten soll und die eigentliche Implementierung sehr zeitaufwendig war, fällt der Algorithmus eher rudimentär aus. Es wurde zwar versucht, ihn zu optimieren, aber im Mittelpunkt dieser Arbeit stand vor allem die technische Realisierung des Programms.

Die Punkte, die bei der Implementierung möglichst berücksichtigt werden sollten, wurden bereits in der „Analyse vorhandener Algorithmen“ auf Seite 7 näher erläutert. Als Eingabe erwartet der Algorithmus den Graphen (im Algorithmus 3 mit der Menge V bezeichnet), und den Weg, der gefunden werden soll (im Algorithmus 3 mit dem Starttripel s bezeichnet). Da es im Allgemeinen wesentlich mehr Knoten als Kanten in einem Graphen gibt, ist die Speicherung des Graphen als Matrix sehr ungeeignet, wie die Matrix des Graphen (Abbildung 11 auf Seite 19) zeigt:

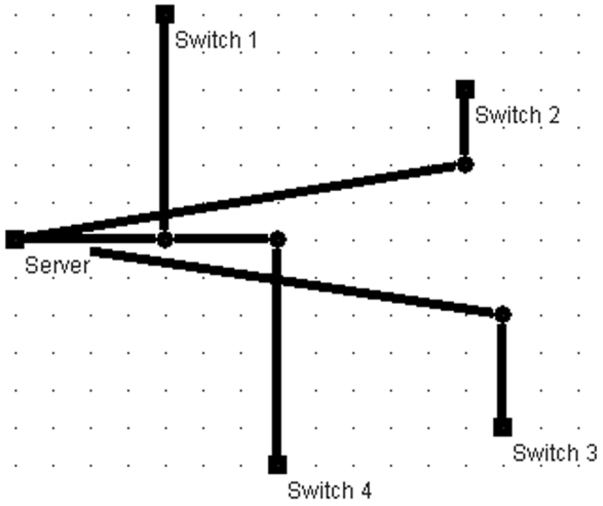

Abbildung 9: Beispielnetzwerk (Abbildung 8 auf Seite 17) als Graph

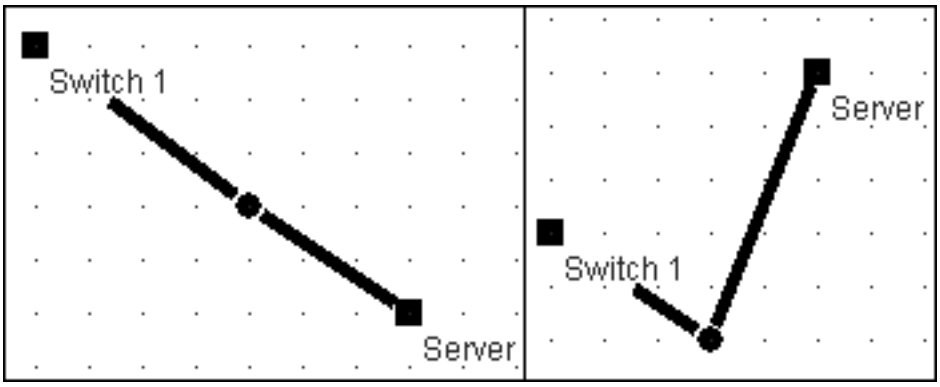

Abbildung 10: Zwei Graphen mit dem gleichen Sachverhalt, dennoch sind sie unterschiedlich. Die Wegstrecke in der linken Abbildung beträgt 5,92 m, die in der Rechten 5,54 m

$$D = \begin{pmatrix} 0 & 2 & 4 & \infty & \infty & \infty & \infty \\ 2 & 0 & \infty & \infty & \infty & \infty & \infty \\ 4 & \infty & 0 & \infty & 1 & \infty & \infty \\ \infty & \infty & \infty & 0 & 2 & \infty & \infty \\ \infty & \infty & 1 & 2 & 0 & 2 & \infty \\ \infty & \infty & \infty & \infty & 2 & 0 & \infty \\ 3 & \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

Zum einen enthält die Matrix sehr viele Einträge, die mit dem ∞-Zeichen gefüllt sind, was bedeutet, dass es zwischen diesen beiden Knoten keine Kante gibt. Des Weiteren fällt auf, dass die Matrix an der Hauptdiagonalen gespiegelt ist, was daran liegt, dass es sich um einen ungerichteten Graphen handelt. Wenn es eine Kante zwischen dem Knoten 1 und 2 gibt, dann auch eine zwischen dem Knoten 2 und 1. Zwar wird der Speicheraufwand nur noch halb so groß, wenn man nur eine Matrixhälfte sichert, sie enthält aber immer noch viele überflüssige Einträge. Deshalb erfolgt die Speicherung des Graphen als eine Menge von Tripeln in Anlehnung an

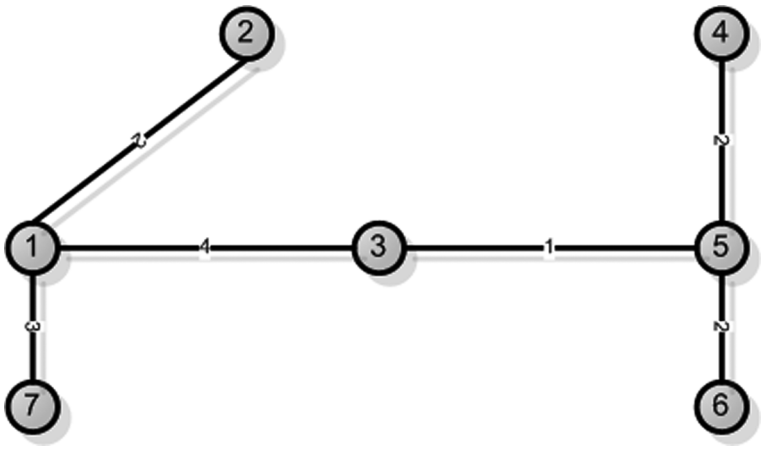

Abbildung 11: Einfacher Graph

[22]. Die Tripel haben die Form:

$$(\text{Startknoten}, \text{Zielknoten}, \text{Kosten})$$

Der dritte Eintrag des Tripels - *Kosten* - kann je nach Einsatz eine andere Bedeutung haben. Er kann die Länge einer Kante beinhalten, die Kapazität einer Kante usw. Nutzt man noch die Eigenschaft aus, dass der Graph ungerichtet ist, so ergibt sich folgende Menge V, wenn man den Graphen wie in Abbildung 11 auf Seite 19 in dieser Tripelschreibweise darstellt:

$$V = \{(1,2,2),(1,7,3),(1,3,4),(3,5,1),(4,5,2),(5,6,2)\}$$

Bei der Implementierung darf man dann nicht vergessen zu beachten, dass eigentlich jedes Tripel doppelt vorkommt, wobei die ersten beiden Einträge miteinander vertauscht sind. Das Starttripel s stellt den Weg dar, für den die kürzeste Wegführung berechnet werden soll. Es hat die Form: $s = (\textit{Startknoten}, \textit{Zielknoten}, \textit{max. Länge})$.

Der implementierte Algorithmus (Pseudokode siehe Seite 20) lehnt sich stark an den Dijkstra-Algorithmus aus [22] an. Er ist nur etwas modifiziert worden, um den besonderen Anforderungen gerecht zu werden.

Zunächst werden die Daten initialisiert. Dann beginnt die eigentliche Suche der kürzesten Weg vom Startknoten aus. Dafür müssen zunächst die kürzesten Wege zu allen anderen Knoten berechnet werden, bis man den Zielknoten erreicht hat. Im schlimmsten Fall ist dies der Knoten, der am weitesten vom Startknoten entfernt ist und somit derjenige, der zuletzt gefunden wird. Im Anschluss muss noch der richtige Weg ermittelt werden, der zum eigentlichen Ziel führt. Dabei wird gleich eine Fehleranalyse gestartet. Wenn ein Weg gefunden worden ist, so wird dieser zurückgegeben. Wenn der gefundene Weg länger ist als die maximal erlaubte Länge, so kommt es zu einer Fehlermeldung. Es kann aber auch passieren, dass kein Weg gefunden wird, weil es sich um keinen zusammenhängenden Graphen handelt. In diesem Fall gibt der Algorithmus eine entsprechende Fehlermeldung zurück.

Die Funktion *minimum()* ermittelt von einer Menge aus Tripeln dasjenige Tripel, welches die geringsten Kosten hat. Es muss beachtet werden, dass die Reihenfolge in S eine wichtige Rolle spielt, da diese die Reihenfolge der zu durchlaufenden Knoten wiedergibt. In der vorliegenden Form berücksichtigt der Algorithmus der Einfachheit halber allerdings nicht, dass jedes Tripel doppelt vorkommt (siehe „Resümee für die Implementierung“ auf Seite 9).

```
Eingabe : Starttripel s, Menge aller Kanten V
Ergebnis : Berechneter kürzester Weg S
Daten : int länge, Tripel such, Tripelmenge D, Tripelliste S

// Initialisierung der Daten
länge = 0
such = (s.start, s.start, 0)
// Alle vom Startknoten aus erreichbaren Knoten ermitteln
für alle v ∈ V tue
    wenn s.von == v.von dann
        D = D ∪ {v}

// Berechnung der kürzesten Wege vom Startknoten aus
solange D.size > 0 oder s.nach != such.nach tue
    such = minimum(D)
    D = D \ {such}
    E = E ∪ {such}
    für alle v ∈ V tue
        wenn such.nach == v.von dann
            für alle d ∈ D tue
                wenn v ∉ E und (v ∉ D oder (v.kosten < d.kosten und v.ziel ==
                d.ziel)) dann
                    D = D ∪ {v}

// Ermittlung des Weges vom Startknoten zum Zielknoten
such = (s.ziel, s.ziel, 0)
solange s.start != getFirst(S).start tue
    für alle e ∈ E tue
        wenn such.von == e.nach dann
            S = putFirst(S, e)
            länge = länge + e.kosten
            such = e
            break

// Auswertung der Berechnung
wenn S.size == 0 dann
    return "Es konnte leider kein Weg ermittelt werden"
wenn länge > s.kosten dann
    return "Der ermittelte Weg ist leider zu lang"
sonst
    return S
```

Algorithmus 3 : Algorithmus zur Berechnung der optimalen Kabelführung

Insgesamt muss der ganze Algorithmus so oft wiederholt werden, wie es zu berechnende Wege gibt. Das Problem, dass die Kanten nur eine bestimmte Kapazität besitzen, ist so gelöst worden, dass es in der Implementierung neben der Menge V noch eine weitere Menge gibt, die die gleichen Tripel wie V enthält, nur das unter Kosten die Kapazität der jeweiligen Kanten vermerkt ist. Nach jedem Durchlauf des Algorithmus wird dann geprüft, über welche Kanten der ermittelte Weg führt und deren Kapazität um eins verringert. Hat eine Kante die Kapazität Null erreicht, so wird diese aus der Menge V gestrichen.

Des Weiteren spielt auch die Reihenfolge der zu ermittelnden Wege, wie das „Resümee für die Implementierung“ auf Seite 9 gezeigt hat, eine wichtige Rolle. Die implementierte Strategie fängt mit der Leitung an, die die geringste maximale Kabellänge hat, danach die nächst größere bis schlussendlich zur größten maximalen Länge. Damit verfolgt die Strategie das Ziel, die kritischsten Wege (die mit der kleinsten Maximallänge) zuerst zu finden und sich dann den weniger kritischen zuzuwenden. Es sei aber darauf hingewiesen, dass diese Strategie nicht immer zum Ziel führen muss, wie man sich an den Abbildungen 4 und 5 auf Seite 10 klarmachen kann. Angenommen, die maximale Länge des Weges C ⇒ B liegt etwas über die der durchgezogenen Linie von Abbildung 5 und die maximale Länge von A ⇒ B ist etwas größer als die Strecke von A ⇒ B, aber geringer als die Länge der gepunkteten Linie. Der Algorithmus berechnet dann zunächst den Weg für C ⇒ B und ermittelt die gestrichelte Linie. Wenn man jetzt annimmt, dass die Kanten eine Kapazität von eins haben, dann lässt sich kein Weg mehr für A ⇒ B ermitteln. Das Problem hat aber eine Lösung, wie man in der Abbildung 4 sehen kann. Eine mögliche Lösung, dass A ⇒ B den gepunkteten Weg nimmt und C ⇒ B den gestrichelten.

3.5 Erläuterung der Konfigurationsdatei

Um das Programm möglichst flexibel zu halten, bietet es diverse Einstellungsmöglichkeiten an, die besonders die Erstellung des Graphen betreffen. Da aber die Standardeinstellungen für 99% der Fälle ausreichend sind und somit Änderungen nur sehr selten vonnöten sind, können diese nicht direkt im Programm manipuliert werden. Stattdessen gibt es eine zentrale Datei, namens Einstellungen.xml, die alle Parameter beinhaltet. Änderungen in dieser Datei werden dann bei einem Neustart des Programms übernommen. Definiert wird die Datei durch die DTD *properties_version_1.dtd* (siehe Listing 3 auf Seite 21). Die Bedeutung der einzelnen

```
<!ELEMENT properties EMPTY>
<!ATTLIST properties
        std_color (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
            12) #REQUIRED
        label_color (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
            12) #REQUIRED
        selec_color (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
            12) #REQUIRED
        std_capacity CDATA #REQUIRED
        std_length CDATA #REQUIRED
        cell_size CDATA #REQUIRED
        grid_gap CDATA #REQUIRED
        port_size CDATA #REQUIRED
        edge_width CDATA #REQUIRED
>
```

Listing 3: properties_version_1.dtd

Parameter können dem Benutzerhandbuch unter „Konfiguration von CaNDeL[2]“ auf Seite 28 entnommen werden.

3.6 Erläuterung des Speicherformats

Um dem Programm die Möglichkeit des Speichern und Ladens zu geben, bedarf es eines geeigneten Formats, welches alle notwendigen Daten aufnehmen kann, um daraus zu einem späteren Zeitpunkt den Graphen rekonstruieren zu können. Es muss die Position der einzelnen Knoten gesichert werden, ebenso etwaige Beschriftungen, die Kanten und deren Kapazität sowie bereits eingelesene PDF- und NDML-Dateien. Aus diesen Anforderungen ist die Beschreibung *save_ version_ 1.dtd* (siehe Listing 4 auf Seite 22) entstanden. *saveData* sichert projektweite Daten, wohingegen *node* die

```
<!ELEMENT saveData (nodes, edges)>
<!ATTLIST saveData
        name CDATA #REQUIRED
        ndml CDATA #IMPLIED
        pdf CDATA #IMPLIED
>

<!ELEMENT nodes (node*)>
<!ELEMENT node EMPTY>
<!ATTLIST node
        id ID #REQUIRED
        xPos CDATA #REQUIRED
        yPos CDATA #REQUIRED
        deviceID CDATA #IMPLIED
>

<!ELEMENT edges (edge*)>
<!ELEMENT edge EMPTY>
<!ATTLIST edge
        startNode IDREF #REQUIRED
        endNode IDREF #REQUIRED
        label CDATA #IMPLIED
>
```

Listing 4: save_version_1.dtd

Eigenschaften jedes einzelnen Knotens und *edge* die Eigenschaften jeder einzelnen Kante aufnimmt. Dabei haben die einzelnen Attribute die folgende Bedeutung:

- *name* ist der Name des Projekts
- *ndml* ist der Pfad zu der zuvor eingelesenen NDML-Datei. Wurde noch keine eingelesen, so entfällt dieser Eintrag
- *pdf* ist der Pfad zu der zuvor eingelesenen PDF-Datei. Wurde noch keine eingelesen, so entfällt dieser Eintrag
- *id* ist die systemweit einmalige ID eines Knotens
- *xPos* ist die X-Position eines Knotens
- *yPos* ist die Y-Position eines Knotens
- *deviceID* ist die ID des Geräts, welches einem Knoten zugewiesen worden ist. Wenn noch keines zugewiesen worden ist, so entfällt dieser Eintrag

[2]Cable Network Design TooL

- *startNode* ist der Startknoten der Kante
- *endNode* ist der Zielknoten der Kante
- *label* ist die Kapazität der Kante, sofern sie von der Standardkapazität abweicht

3.7 Verwendete Frameworks

Neben der Berechnung der kürzesten Wege muss das Programm noch weitere Funktionen bieten. Es müssen PDF-Dateien dargestellt, XML-Dateien eingelesen und Graphen gezeichnet werden. Recherchen haben ergeben, dass bereits viele Frameworks existieren, die diese gewünschten Aufgaben erfüllen.

Für die Darstellung der PDF-Dateien habe ich mich für das Framework JPedal entschieden [15], welches in zwei Varianten erhältlich ist. Es gibt eine kommerzielle Version und eine freie Version, die unter der GPL Lizenz [11] steht. Diese ist zwar etwas älter als die kommerzielle Version, aber für die Zwecke des Programms vollkommen ausreichend.

Für die Erstellung des Graphen habe ich das Framework JGraph [19] gewählt. Auch dieses Framework liegt sowohl als kommerzielle Variante, als auch als freie Variante, die unter der LGPL [12] steht, vor. JGraph stellt unter anderem alle Funktionen zur Verfügung, die zur komfortablen Erstellung und Editierung eines Graphen notwendig sind.

Um XML-Dateien einfach einlesen und schreiben zu können, kommt das Framework JDom [17] zum Einsatz. Dieses liegt unter der Apache License [10], die vergleichbar mit der LGPL ist. Es ist recht einfach in der Handhabung und kann zum Beispiel XML-Dateien auch validierend einlesen und sie so auf Fehler analysieren.

3.8 Die Programmstruktur

Das Programm setzt sich aus den Paketen `dialog`, `graphComponents`, `networkData`, `properties` und `solver` zusammen, wobei das Paket `graphComponents` noch die Pakete `views` und `handler` beinhaltet (graphische Darstellung siehe Abbildung 12 auf Seite VI).

Die `main()`-Funktion des Programms befindet sich in der Klasse `Main` und startet nur den `Coordinator`. Dieser übernimmt die komplette Verwaltung des Programms und vermittelt zwischen den einzelnen Paketen.

Das Paket `properties` verwaltet die Daten zur Konfiguration des Programms, wie es unter „Erläuterung der Konfigurationsdatei properties_version_1.dtd“ auf Seite 21 geschildert ist. Das Paket `networkData` ist die Mittelschicht zwischen den Datenstrukturen des Graphen und der Darstellung des Graphen. Zudem verwaltet es die Geräte, die aus der NDML-Beschreibung ausgelesen werden. Das Paket `dialog` beinhaltet Dialoge, um mit dem Nutzer zu kommunizieren. Das Paket `solver` implementiert den Algorithmus, wie er unter „Optimale Wege berechnen“ auf Seite 17 skizziert worden ist. Um den Algorithmus möglichst gut austauschbar zu machen, sind die benötigten Datenstrukturen und Methoden in Interfaces (siehe Listing 5, 6 und 7 auf Seite 24, 25 und 26) definiert worden.

Zudem ist die Methode `solveTheProblem()` in `ISolver` mit der Signatur `List <IReturnData> solveTheProblem (Object graph, Object edgesWanted,`

```
public interface ISolver
{
        public final static int TO_LONG_ERROR = -1;
        public final static int NO_WAY_ERROR = -2;
        public final static int NO_ERROR = 1;

        /**
         * Berechnet die kuerzesten Wege in einem Graphen
         *
         * @param graph Graph, in dem die kuerzesten Wege berechnet
         *      werden sollen
         * @param edgesWanted Wege, üfr die der kuerzeste Verlauf
         *      ermittelt werden soll
         * @param edgesCapacity Die einzelnen Kapazitaeten der Kanten
         *
         * @return Liste der ermittelten kuerzesten Wege
         */
        public List<IReturnData> solveTheProblem(Object graph,
                Object edgesWanted, Object edgesCapacity);
}
```

Listing 5: ISolver.java

`Object edgesCount)` sehr allgemein gehalten. Es wird lediglich festgelegt, dass der Methode der Graph, die gesuchten Wege und die Kapazitäten der Kanten übergeben werden müssen. Die Art und Weise, wie dies geschieht, bleibt dem Programmierer vorbehalten. Implementiert wurde eine Variante in Anlehnung an „Optimale Wege berechnen“ auf Seite 17. Die Tripel werden in `ITriple` definiert. Die Rückgabe der berechneten Wege erfolgt über die Struktur `IReturnData`.

Das Paket `graphComponents` enthält nur Klassen, die von dem Framework JGraph erben und an die Bedürfnisse des Programms angepasst sind. Als Unterpakete enthält es noch die Pakete `handler` und `views`.

4 Benutzerhandbuch

Dieser Abschnitt erläutert, wie man das Programm CaNDeL starten und nutzen kann. Im Unterordner *data* befinden sich unter anderem drei Beispieldateien. Die Datei *Netzwerk.xml* ist eine NDML-Beschreibung eines Netzwerks mit einem Server und fünf Switches. Die Datei *Beispiel.xml* enthält einen Beispielgraphen und die Datei Grundriss.pdf enthält ein Beispiel für einen Grundriss.

4.1 Start des Programms

Öffnet man den Ordner vom Programm CaNDeL (Abbildung 13 auf Seite VII), so enthält dieser drei Einträge. Der Ordner *pics* enthält Graphiken, die vom Programm selbst genutzt werden. Der Ordner *data* ist das Arbeitsverzeichnis des Programms und die JAR-Datei das eigentliche Programm. Zum Starten des Programms reicht häufig ein Doppelklick auf die Datei aus, vorausgesetzt, dass Java mindestens in der Version 1.5 installiert ist. Falls nicht, so kann man es unter [16] herunterladen.

```
public interface IReturnData
{
        /**
         * Liste von ITriples, die die durchlaufenen Kanten enthalten.
         */
        public List getList();

        /**
         * Laenge des berechneten Wegs.
         */
        public double getLength();

        /**
         * Fehlercode, der waehrend der Berechnung aufgetreten ist.
         * @see ISolver#NO_ERROR
         * @see ISolver#NO_WAY_ERROR
         * @see ISolver#TO_LONG_ERROR
         */
        public int getErrorCode();

        /**
         * Das ITriple beinhaltet den Weg, der berechnet werden sollte.
         */
        public ITriple getTriple();
}
```

Listing 6: IReturnData.java

4.2 Knoten einfügen

Die Knoten des Graphen kann man platzieren, indem man in der Symbolleiste auf das Knotensymbol (Abbildung 14 auf Seite VII) klickt. Führt man danach den Mauszeiger in die Zeichenfläche, so ändert sich sein Aussehen und mit einem Linksklick wird ein Knoten eingefügt (Abbildung 15 auf Seite VII). Dieser wird automatisch an dem Raster ausgerichtet. Stellt man während des Klicken fest, dass man die falsche Position gewählt hat, so kann man bei noch gedrückter linker Maustaste die Knoten verschieben.

4.3 Kanten einfügen

Mit einem Klick auf das Maussymbol (Abbildung 16 auf Seite VII) in der Symbolleiste kann man in den entsprechenden Modus wechseln. Um die Kante zu zeichnen, bewegt man die Maus über einen Knoten. Der Mauszeiger verwandelt sich in ein Handsymbol. Zum Einfügen einer neuen Kante bewegt man mit gedrückter linker Maustaste den Zeiger zum Zielknoten und lässt die Taste dort wieder los. Dort die Maustaste wieder loslassen und die Kante wird eingefügt (Abbildung 17 auf Seite VIII).

Knoten können verschoben werden, in dem man mit der Maus darüber fährt, bis sich der Mauszeiger in ein Pfeilkreuz ändert. Mit gedrückter linker Maustaste den Knoten an die gewünschte Stelle ziehen und die Taste wieder loslassen. (Abbildung 18 auf Seite VIII).

Es können auch mehrere Knoten gleichzeitig verschoben werden, indem man mehrere Knoten selektiert und dann einen von ihnen verschiebt. Mehrere Knoten kann man selektieren, indem man mit der Maus an eine beliebige Stelle klickt und mit gedrückter linker Maustaste einen Kasten aufzieht (Abbildung 19 auf Seite VIII).

```
public interface ITriple extends Comparable, Cloneable
{
        /**
         * Ermittelt, ob zwei Tripel gleich sind. Da es sich bei dem
         * Graphen der Kabelkanaele um einen ungerichteten Graphen
         * handelt, spielt es keine Rolle, ob man (Startknoten,
         * Zielknoten, Kosten) oder (Zielknoten, Startknoten, Kosten)
         * schreibt. Deshalb werden diese beiden Tripel als gleich
         * angesehen.
         *
         * @param obj Das Triple, mit dem verglichen werden soll
         * @return true, wenn gilt Triple1 = (x1, y1, z1) und
         *      Triple2 = (x2, y2, z2) mit (x1 == x2 und y1 == y2)
         *      oder (x1 == y2 und y1 == x2), ansonsten false
         */
        public boolean equals(Object obj);

        public Object getSource();

        public Object getTarget();

        public double getCost();

        public void setSource(Object source);

        public void setTarget(Object target);

        public void setCost(double i);

        public ITriple clone();
}
```

Listing 7: ITriple.java

4.4 Knoten löschen

Es können jederzeit selektierte Knoten durch Drücken der Entf-Taste gelöscht werden. Wenn dieser Knoten bereits mit Kanten verbunden war, so werden diese ebenfalls gelöscht.

4.5 PDF-Datei einlesen

Eine PDF-Datei kann man einlesen, indem man in der Symbolleiste auf das PDF-Icon klickt (Abbildung 20 auf Seite VIII). Daraufhin öffnet sich ein Fenster, in dem man die gewünschte PDF-Datei auswählen kann. Hat man die gewünschte Datei gefunden, wird man dazu aufgefordert den Maßstab des Grundrisses einzugeben (Abbildung 21 auf Seite IX), damit die Längen der Kanten richtig berechnet werden können. Anschließend wird die Datei in den Hintergrund gezeichnet.

4.6 NDML einlesen und Geräte zuweisen

Eine NDML-Datei kann man einlesen, indem man auf das NDML-Symbol in der Symbolleiste klickt (Abbildung 22 auf Seite IX). Daraufhin öffnet sich ein Fenster, in dem man die gewünschte NDML-Datei auswählen kann. Danach stehen die Daten der NDML-Datei im Programm zur Verfügung.

4.7 Geräte und Beschriftungen einfügen

Standardmäßig haben die Kanäle eine Kapazität von fünf. Soll diese individuell geändert werden, so kann man die Kapazität einer Kante ändern, indem man auf diese doppelklickt und die gewünschte Kapazität einträgt (Abbildung 23 auf Seite IX). Die Eingabe mit Enter bestätigt.

Den einzelnen Knoten kann man diejenigen Geräte zuweisen, die in der NDML definiert sind. Dazu muss als erstes eine NDML-Datei eingelesen werden. Danach einfach auf einen Knoten doppelklicken und in der sich öffnenden Liste das gewünschte Gerät auswählen. Die Beschriftung wird daraufhin in den Graphen eingefügt und das Symbol ändert sich zu einem Viereck (Abbildung 24 auf Seite X). Die Zuweisung kann jederzeit wieder entfernt werden, in dem man den Eintrag „Zuweisung entfernen“ in der Liste auswählt.

4.8 Berechnung der kürzesten Wege

Um die Berechnung der kürzesten Wege zu starten, muss man auf das Rechensymbol in der Symbolleiste klicken (Abbildung 25 auf Seite X). Falls zuvor noch kein PDF eingelesen worden ist, wird man jetzt aufgefordert, den Maßstab des Graphen anzugeben. Anschließend beginnt die Berechnung. Ist diese Beendet, so öffnet sich ein zweites Fenster, in dem das Ergebnis in Form einer Tabelle dargstellt wird (Abbildung 26 auf Seite XI).

Für jeden Weg, der in der NDML-Datei beschrieben ist, wird ein kürzester Pfad durch den Graphen gesucht, sofern die entsprechenden Geräte auch zugewiesen worden sind. Die erste beiden Spalten der Tabelle geben die Endpunkte des Weges wieder, für die der kürzeste Pfad berechnet werden sollte. Die dritte Spalte enthält die Knoten, die man überqueren muss, um den kürzesten Weg zu gehen oder eine entsprechende Fehlermeldung, wenn kein Pfad gefunden werden konnte, entweder weil der Pfad zu lang ist oder weil es keine Möglichkeit gibt, die beiden Endpunkte zu verbinden. Ist eine direkte Verbindung wie zwischen dem Server und dem Switch 5 (Abbildung 26 auf Seite XI) möglich, so gibt es eine entsprechende Ausgabe. Damit der Weg leicht nachvollzogen werden kann, werden die zu durchquerenden Knoten mit einer entsprechenden Nummer versehen. Um Abschätzen zu können, wie viel Reserve das Kabel noch bietet, ist in der vierten und fünften Spalte aufgelistet, wie lang das Kabel maximal sein darf und wie lang der berechnete Pfad ist.

Über den Button *drucken* kann man die Ergebnistabelle ausdrucken. Der Button *schließen* schließt das Ergebnisfenster. Um sich das Ergebnis zu einem späteren Zeitpunkt nochmals anzeigen zu lassen, einfach in der Symbolleiste auf das Ergebnissymbol klicken (Abbildung 27 auf Seite XI). ACHTUNG: Es wird keine neue Berechnung gestartet, sondern nur das zuvor berechnete Ergebnis angezeigt.

4.9 Ausdrucken

Das Ergebnis der Berechnung kann über das Ergebnisfenster (siehe „Das Ergebnis einer Berechnung“ auf Seite 27) gedruckt werden. Der erstellte Graph kann mitsamt Hintergrundbild über das Drucksymbol in der Symbolleiste gedruckt werden (Abbildung 28 auf Seite XI).

4.10 Das Menü

Alle zuvor beschriebenen Funktionen lassen sich auch über das Menü ausführen. Zusätzlich bietet das Menü „Datei“ an, einen neuen Graphen zu erstellen. Hierbei werden alle bis dahin noch nicht gespeicherten Änderungen verworfen. Man kann außerdem die Seiteneinstellung für die Druckdatei vornehmen und bspw. die Seitenränder neu einstellen. Des Weiteren lassen sich dort Einträge zum Laden und Speichern finden. Wichtig beim Laden und Speichern ist, dass sich die Datei *save_version_1.dtd* in dem gleichen Ordner, wie die zu speichernde/ladende Datei, befindet.

Das Menü „Über“ öffnet ein Fenster mit Informationen über CaNDeL.

4.11 Konfiguration von CaNDeL

Die wichtigsten Einstellungen in CaNDeL lassen sich individuell auf die eigenen Bedürfnisse abstimmen. Um Änderungen vorzunehmen muss man die Datei „Einstellungen.xml“ im Unterordner *data* öffnen und diese nach seinen Wünschen editieren. Die verschiedenen Parameter im einzelnen:

- *std_color* legt die Farbe fest, in der die Knoten und Kanten gezeichnet werden
- *label_color* legt die Farbe fest, in der die Beschriftungen der Knoten und Kanten geschrieben werden
- *selec_color* bestimmt die Farbe, wenn Knoten und/oder Kanten selektiert werden. Die etwaigen zugehörigen Beschriftungen nehmen ebenso diese Farbe an
- *std_capacity* legt fest, wie groß die Standardkapazität der Kanäle sein soll, wenn diese nicht explizit angeben wird
- *std_length* legt fest, wie lang die maximale Länge einer Leitung ist, wenn das optionale Attribut `length` in der NDML-Beschreibung des Netzwerks nicht gesetzt ist
- *cell_size* bestimmt die Größe der einzelnen Knoten
- *grid_gap* definiert den Abstand der einzelnen Grid-Punkte
- *port_size* legt die Größe der Ports der Knoten fest. Dieser Wert sollte immer kleiner als *cell_size* sein
- *edge_width* gibt die Breite der Kanten vor

Die Farben der ersten drei Parameter werden mittels Zahlen angegeben, wie sie in der Tabelle 2 auf Seite 29 angegeben sind.

5 Resümee

Die Aufgabe dieser Belegarbeit bestand darin, ein Programm zu schreiben, welches eine optimale Verlegung von Leitungen in einem System aus Kabelkanälen berechnet. Dazu mussten in der kürze der Zeit mögliche technische Realisierungen ausgelotet und die am Erfolg versprechenste implementiert werden. Entstanden ist

Wert	Farbe
0	schwarz
1	blau
2	rot
3	cyan
4	dunkelgrau
5	grau
6	grün
7	hellgrau
8	magenta
9	orange
10	pink
11	weiß
12	gelb

Tabelle 2: Farbzuordnung in der properties_version_1.dtd

dabei das vorliegende Programm CaNDeL, welches eine Möglichkeit der Umsetzung prototypisch demonstrieren soll.

Zunächst musste man sich in das Thema der Kabelverlegung einarbeiten. Das heißt, es galt sich mit gesetzlichen Bestimmungen und geltenden Richtlinien vertraut zu machen, sowie der optimalen Verlegung von Kabeln in einem Kanalsystem vertraut zu machen. Mathematisch gesehen ist dies ein Problem der optimalen Wegfindung in einem Graphen, erwähnt seien dazu an dieser Stelle nur kurz die Algorithmen „Bellman-Ford“ und „Dijkstra“.

Des Weiteren musste eine Programmstruktur entworfen werden, die möglichst gut wartbar und erweiterungsfähig ist. Dazu gehört auch die Suche nach bereits vorhanden Programmen und Frameworks, um diese entweder selber einsetzen zu können oder um diese zu analysieren, um Lösungen für eigene Probleme zu finden. Ebenso wichtig wie eine flexible Programmstruktur ist eine gute Benutzerschnittstelle, damit der Anwender des Programms möglichst schnell und ohne größere Probleme mit dem Programm arbeiten kann. Das Hauptaugenmerk lag hierbei auf der Erstellung des Kabelkanalnetzwerks und der Ausgabe des berechneten Ergebnisses. Dieses muss auch in der Form zur Verfügung stehen, dass sie ein Elektroinstallateur lesen kann, um mit Hilfe dessen die Kabel verlegen zu können. Aber auch die Anpassungsfähigkeit des Programms an die eigenen Anforderungen spielt eine wichtige Rolle.

Die bisherige Beschreibung des Netzwerks durch die *ndml_version_1.1.dtd* konnte nicht genutzt werden, um das System der Kabelkanäle zu erfassen, da die Struktur dafür nicht ausgelegt ist. So musste eine neue Struktur entwickelt werden, die diese Aufgabe erfüllen kann. Die Datei *save_version_1.dtd* ist ein Beispiel, wie so etwas aussehen kann. Für die Zukunft bleibt zu überlegen, ob nicht die beiden Strukturen zu einer zusammengeführt werden sollten bzw. die Kanalstruktur in die NDML aufgenommen werden soll (siehe NDML Version 2.0 und 3.0 [14]). Dies hätte den Vorteil, dass es keine zweigeteilte Datenhaltung mehr gibt und alle projektrelevanten Informationen an einer zentralen Stelle gelagert werden. Allerdings könnte es auch zum Nachteil geraten, wenn die anderen Programmteile nie auf diese Informationen zugreifen, und sich dadurch die Struktur unnötigerweise „aufbläht“. Je nach Realisierung der Verschmelzung wären keine oder nur sehr geringe Anpassungen in den übrigen Programmteilen von CANDY notwendig und auch bei CaNDeL wären

die Anpassungen schnell durchgeführt.

Um den Code möglichst leicht wartbar zu gestalten, sollte dieser gut strukturiert und kommentiert sein. Deswegen gibt es für jede Methode eine Erläuterung in der zugehörigen JavaDoc. Falls diese nicht ausreichend ist, befinden sich auch noch viele Kommentare im Code selbst, in denen näher auf implementierungsspezifische Aspekte eingegangen wird.

In diesem Beleg wurde die Entwicklung des Programms CaNDeL erläutert, mit dessen Hilfe man ein System aus Kabelkanälen erstellen und anschließend die optimalen Wege berechnen lassen kann. Allerdings ist der Begriff „optimal“ in diesem Zusammenhang ein schwieriger Begriff ist. Ein Weg in einem Graphen ist optimal, wenn dieser die kürzeste Verbindung zwischen zwei Knoten darstellt. So stellt sich nun die Frage, wann zwei oder mehr Wege in einem Graphen optimal sind. Wenn der eine kurz ist und der andere Weg dafür länger ist oder wenn beide Wege annähernd gleich lang sind? In CaNDeL ist das Problem derart gelöst, dass zuerst der Weg für die Leitung berechnet wird, welche die kleinste Maximallänge hat, anschließen diejenige mit der zweitkleinsten usw. Eine optimale Lösung ist dann gefunden, wenn für jeden Leitung ein Weg gefunden werden konnte. Allerdings führt diese Herangehensweise nicht immer zu einer optimalen Lösung, wie schon in „Optimale Wege berechnen“ auf Seite 17 festgestellt worden ist.

Da diese Art von Algorithmen schon relativ komplex ist (meist $\mathcal{O}(n^2)$ und aufwärts) bedürfen sie einer genauen Analyse und effizienten Implementierung. Der in CaNDeL implementierte Algorithmus erfüllt seine Aufgabe, bietet aber noch genügend Raum für Verbesserungen hinsichtlich des Ablaufs des Algorithmus und seiner Implementierung. Aufgrund der mangelnden Zeit waren diese leider nicht zu realisieren. Denkbar wäre auch ein Algorithmus, der einen ganz anderen Ansatz verfolgt. Seine Integration in das Programm sollte aufgrund der sehr flexiblen Schnittstelle keine größeren Probleme bereiten.

Schlussendlich bleibt zu überlegen, wie sinnvoll einer Verschmelzung der beiden Strukturen *ndml_ version_ 1.1.dtd* und *save_ version_ 1.dtd* und somit eine Integration der Kabekanalnetzdaten in die NDML-Bescheibung sinnvoll ist. Damit einhergehen würde auch, dass die Definition der *save_ version_ 1.dtd* in ein XML-Schema überführt werden müsste. Dies hätte zusätzlich den Vorteil, dass sich die Typangaben feiner spezifizieren lassen. Auch sollte das Programm CaNDeL noch mehr in die CANDY-Umgebung integriert werden, was aufgrund der Zeit und des deshalb eher prototypischen Charakters des Programms nicht möglich war.

Anhang

Glossar

Apache License Lizenzmodell für freie Software [10], die von der Apache Group, die unter anderem den Apache Webserver entwickelt haben, entwickelt worden ist

CaNDeL *Cable Network Design TooL* Der Name des Programms, welches im Rahmen dieser Belegarbeit entstanden ist

CANDY *Computer Aided Network Design UtilitY* Hierbei handelt es sich um ein Projekt zur Erstellung einer Software im Umfeld der Netzwerkprojektierung [18]

Designer Ist der Nutzer des Programms, der ein Kabelkanalnetzwerk designt

DPI *Dots per Inch* Typische Einheit zur Angabe der Auflösung

DTD *Document Type Definition* Legt die Struktur eines XML-Dokuments fest [3]

GPL *GNU General Public License* Lizenzmodell für freie Software [11]

JavaDoc Automatisch aus dem Java-Quelltext generierte Dokumentation, die auf Kommentaren bestimmter Syntax beruht

LAN *Local Area Network*

NDML *Network Design Markup Language* XML-Dialekt, der von CANDY genutzt wird

LGPL *GNU Lesser General Public License* Lizenzmodell für freie Software [12], weniger restriktiv als GPL

PDF *Portable Document Format* Plattformübergreifendes Dateiformat für druckbare Dokumente, welches von Adobe Systems entwickelt worden ist [8]

Waagerechte Verkabelung Betrachtet man ein Haus bzw. Grundriss von oben und legt dann Leitungen, handelt es sich um eine waagerechte Verkabelung

WLAN *Wireless LAN*

XML *Extensible Markup Language* Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur [6]

XML Schema siehe XSD

XSD *XML-Schema-Definition* Komplexe Sprache zur Beschreibung eines XML-Typsystems [9]

Literatur

[1] Algorithmus von Dijkstra. http://de.wikipedia.org/ wiki/Dijkstras_Algorithmus.

[2] Bellman-Ford-Algorithmus. http://de.wikipedia.org/ wiki/Bellman-Ford-Algorithmus.

[3] Document Type Definition. http://de.wikipedia.org/ wiki/DTD.

[4] Ethernet. http://de.wikipedia.org/wiki/Ethernet.

[5] Ethernet. www.koehler-ks.de/Ethernet.html.

[6] Extensible Markup Language. http://de.wikipedia.org/ wiki/Xml.

[7] Optimalitätsprinzip von Bellman. http://de.wikipedia.org/ wiki/Optimalit

[8] Portable Document Format. http://de.wikipedia.org/ wiki/Pdf.

[9] XML-Schema-Definition. http://de.wikipedia.org/ wiki/XSD.

[10] Apache Software Foundation. Apache License. Internet. www.apache.org/licenses/LICENSE-2.0.

[11] Free Software Foundation. General Public License. Internet. www.gnu.org/licenses/gpl.html.

[12] Free Software Foundation. Lesser General Public License. Internet. www.gnu.org/copyleft/lesser.html.

[13] Clark Helwig. Möglichkeiten und Grenzen des vollautomatischen Entwurfs von Local-Area-Rechnernetzen. Hauptseminar, Fakultät Informatik, TU Dresden, 2004. www.rn.inf.tu-dresden.de/scripts_lsrn/Lehre/candy/vortraege/Helwig_Verkabelung.pdf.

[14] Robert Hänel. Optimierung der Fachsprache NDML mit Abbildung in eine XML-basierte Datenbank. Diplomarbeit, Fakultät Informatik, TU Dresden, 2005. www.rn.inf.tu-dresden.de/scripts_lsrn/Lehre/candy/Haenel/Diplomarbeit.pdf.

[15] IDRsolutions. JPedal Open Source. Internet. www.jpedal.org/gpl.html.

[16] Sun Microsystems Inc. Java Software. Internet. www.java.com/de.

[17] JDOM™. JDom. Internet. www.jdom.org.

[18] TU Dresden Lehrstuhl für Rechnernetze, Fakultät Informatik. Computer Aided Network Design UtilitY. www.rn.inf.tu-dresden.de/scripts_lsrn/Lehre/candy.

[19] JGraph Ltd. JGraph. Internet. www.jgraph.com.

[20] Klaus-Jürgen Schneider. *Bautabellen für Architekten*. Werner Verlag, Neuwied, 16. edition, August 2004.

[21] unitherm. Gesetzliche Grundlagen. Internet. www.unitherm-online.de/wisspool/gesetze/index.php.

[22] Prof. H. Vogler. *Algorithmen, Datenstrukturen und Programmierung*, vorlesungsmaterial Graphalgorithmen, page 141ff. September 2005. www.orchid.inf.tu-dresden.de/gdp/lehre.

Eigenständigkeitserklärung

Hiermit erkläre ich, Philipp Bönisch, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Hilfsmittel verwendet und die Stellen, die anderen Werken dem Wortlaut nach entnommen sind, mit Quellenangaben kenntlich gemacht habe.

(Philipp Bönisch)

Dresden, den 31. Januar 2006

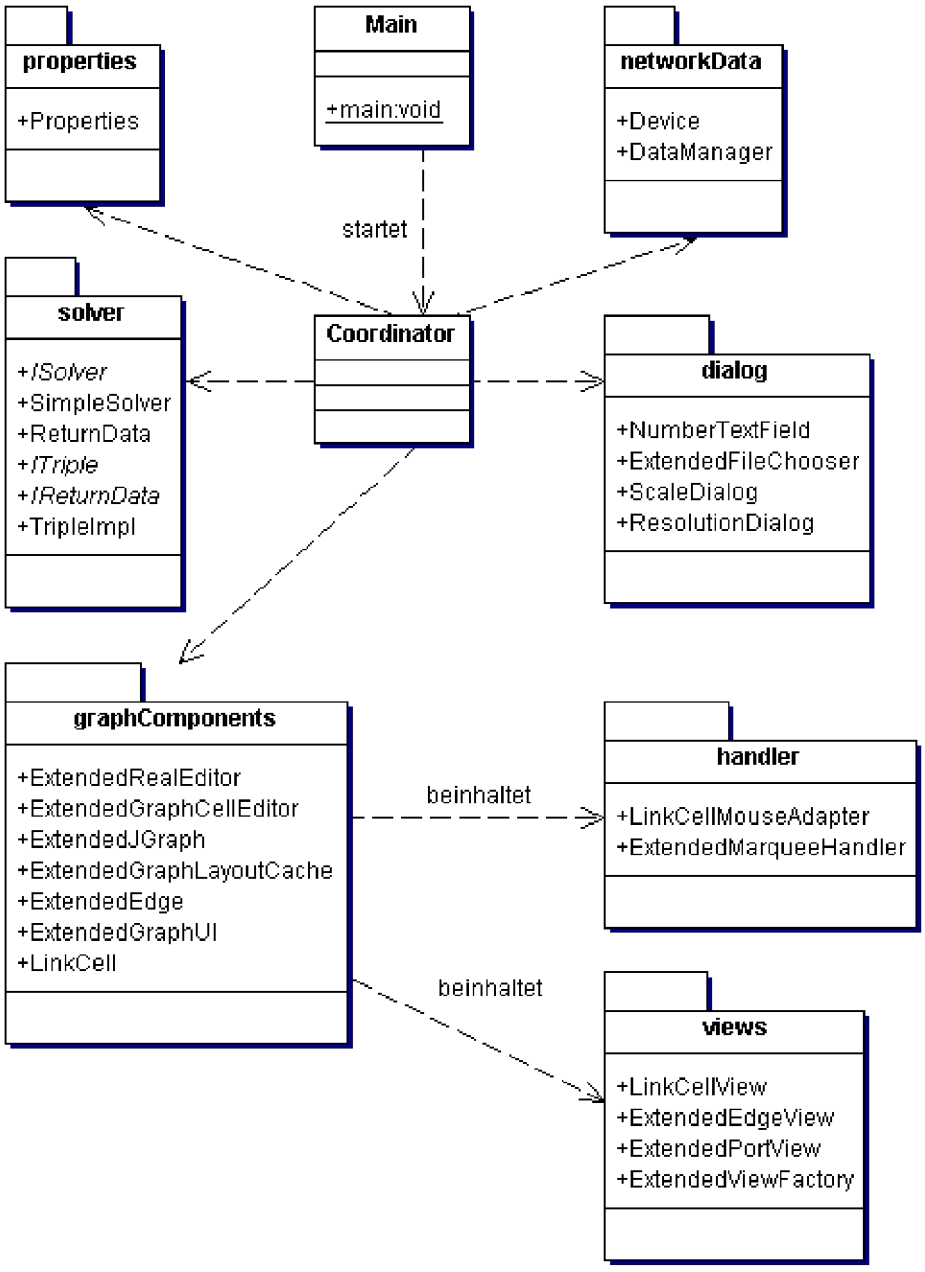

Abbildung 12: Paketstruktur des Programms

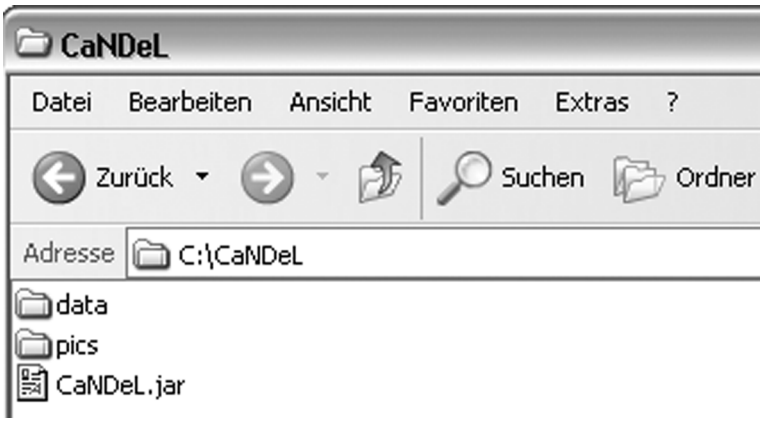

Abbildung 13: Inhalt des Ordners CaNDeL

Abbildung 14: Knoten zeichnen auswählen

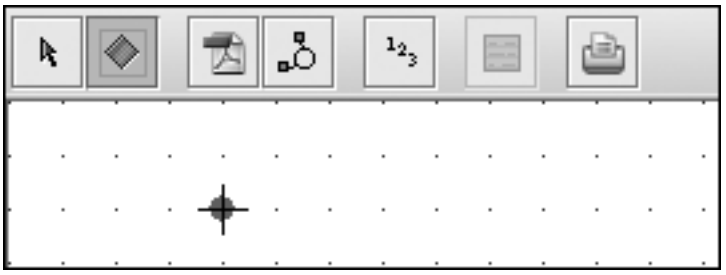

Abbildung 15: Knoten gezeichnet

Abbildung 16: Kanten einfügen auswählen

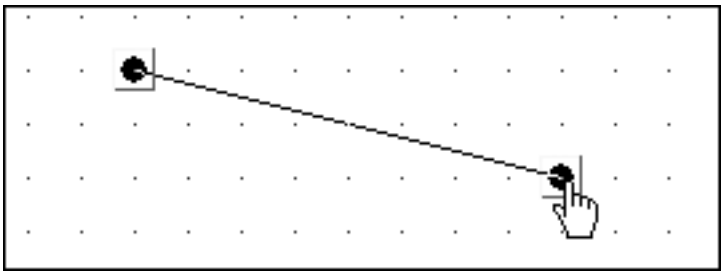

Abbildung 17: Neue Kante hinzugefügt

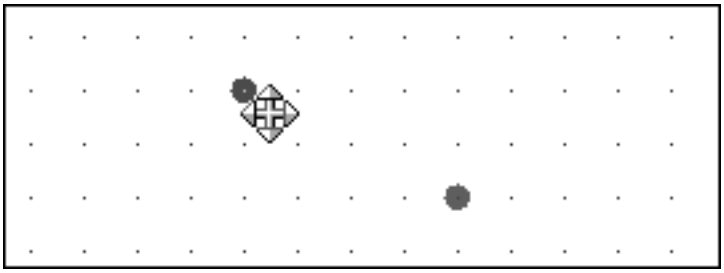

Abbildung 18: Verschieben eines Knoten

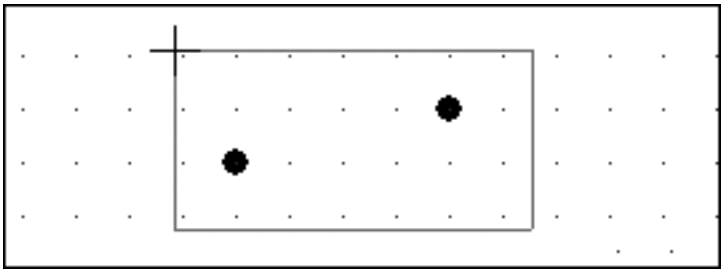

Abbildung 19: Mehrere Knoten selektieren

Abbildung 20: PDF einlesen wählen

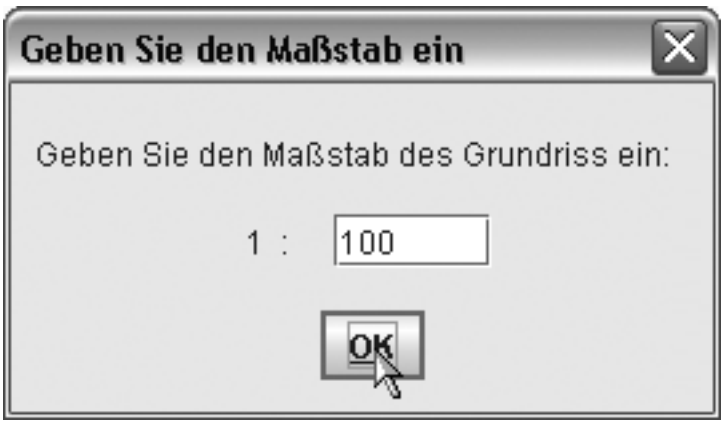

Abbildung 21: Eingabefenster für den Maßstab

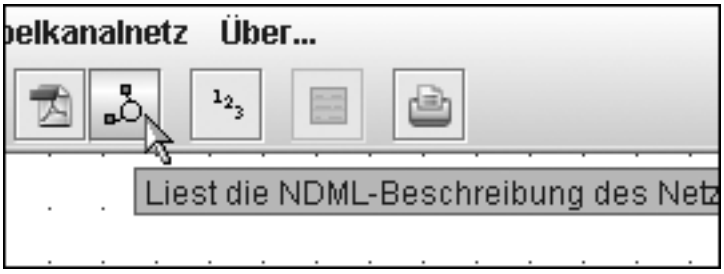

Abbildung 22: NDML einlesen wählen

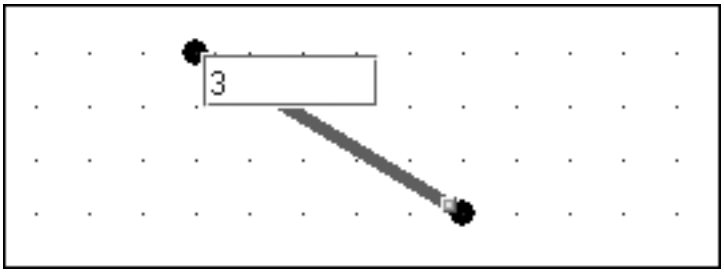

Abbildung 23: Kapazität einer Kante festlegen

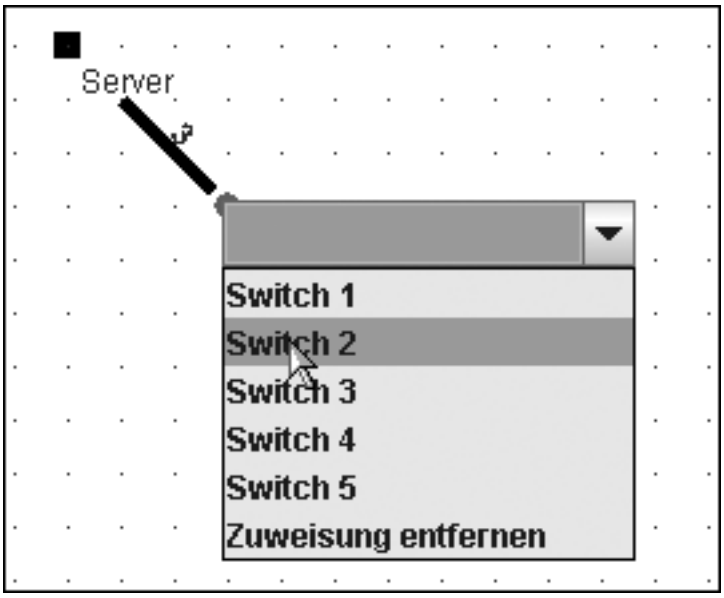

Abbildung 24: Knoten ein Gerät zuweisen

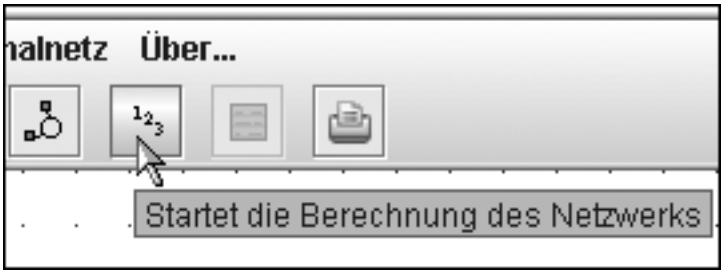

Abbildung 25: Berechnung der kürzesten Wege starten

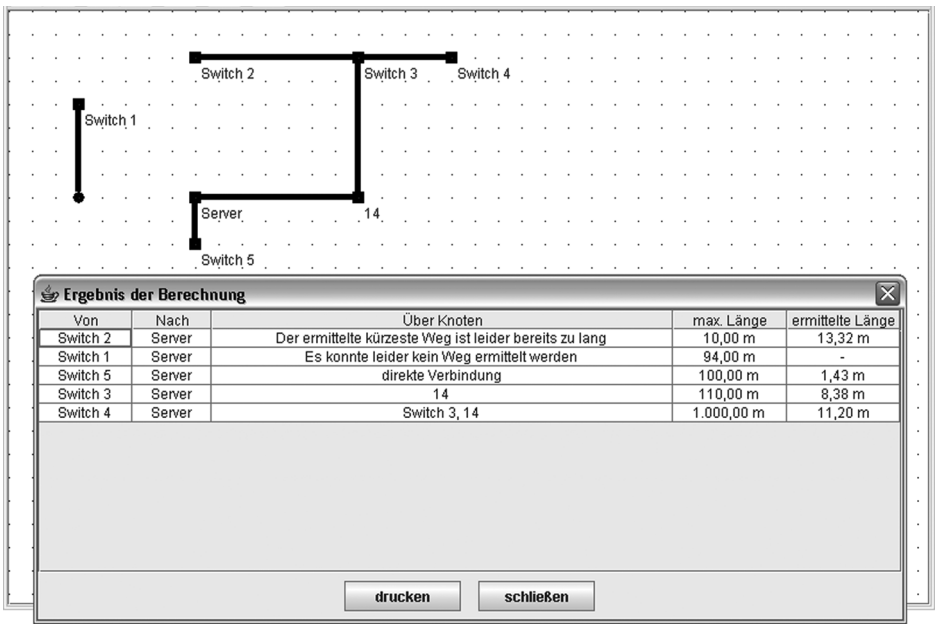
Switch 2
Switch 3
Switch 4
Switch 1
Server
14
Switch 5
Ergebnis der Berechnung
Von | Nach | Über Knoten | max. Länge | ermittelte Länge
Switch 2 | Server | Der ermittelte kürzeste Weg ist leider bereits zu lang | 10,00 m | 13,32 m
Switch 1 | Server | Es konnte leider kein Weg ermittelt werden | 94,00 m | -
Switch 5 | Server | direkte Verbindung | 100,00 m | 1,43 m
Switch 3 | Server | 14 | 110,00 m | 8,38 m
Switch 4 | Server | Switch 3, 14 | 1.000,00 m | 11,20 m
drucken
schließen

Abbildung 26: Das Ergebnis einer Berechnung

Abbildung 27: Zuvor berechnetes Ergebnis nochmals anzeigen

Abbildung 28: Graph ausdrucken